パソコンプログラマのためのクリエイティブマガジン

平成3年4月2日第三種郵便物認可　1991年10月1日発行（毎月1回18日）第1巻第10号通巻10号

月刊**プログラマーズ・ページ**

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1991 10 OCTOBER

定価980円

特集

286/386 メモリ活用プログラミング

MS-DOSからプロテクトメモリを利用するために

付録:5"2HDフロッピーディスク

本誌掲載プログラム／綴じ込み付録テキストファイル（EMSファンクションリクエスト（INT 67H）一覧）／クイズ・ランダムスクリーン／解読ツール／NEC独自文字→JIS定義文字変換フィルタDENEC

SE
Shoeisha Co., Ltd.

BORLAND
The Leader in Object-Oriented Programming

問：Windows™ アプリケーション開発に最も有効なコンパイラは、何ですか？

（解答はページ下をご覧下さい）

(Inactive C:¥WORK¥BORLAND.EXE)

	従来の開発環境	？
Cコンパイラ	¥ 98,000	¥ 98,000
C++コンパイラ	なし	含む
Windows AP作成ツール	¥ 98,000	含む
アセンブラ	¥ 40,000	含む
合計	¥236,000	¥ 98,000

※上記の価格には、消費税は含まれていません。

パソコン上に、視覚的で快適な操作環境を提供するWindows 3.0。そこで利用するアプリケーション作成のためのコンパイラは、現在のものでベストといえるでしょうか？機能、スピード、価格…。これらの面で理想的ともいえるコンパイラが、登場しました。「BORLAND C++」です。BORLAND C++は、Windowsアプリケーションを開発、デバックするC/C++コンパイラとツールを標準で装備しています。

■Windowsアプリケーション開発をサポート

BORLAND C++は、Windowsプログラムを完全にサポート。高速かつ高性能な処理を実現しています。MDI(Multiple Document Interface)やDLLs(Dynamic Link Libraries)といったようなWindowsの機能を使って、大規模なプロテクトモードアプリケーションの開発が可能になります。さらに、MicrosoftのリソースコンパイラとWINDOWS.H、Resource Work Shop(日本語版)が標準装備されていますので、別途の開発キット(SDK)は不要です。

■ANSI C／AT＆T C++両用コンパイラ

BORLAND C++は、CとC++両方のコンパイラとして利用できます。さらに、再コンパイルの速度を大幅に向上させる「コンパイル済みヘッダ」機能をサポート。MS-DOSの640KBというメモリの制限を破るオーバーレイマネージャVROOMの利用や、Turbo Driveによってプロテクトモードで動作する統合開発環境での大規模なプログラムの作成も可能です。

■Programmer's Platform

Programmer's Platformは、好きなツールを組み込む機能を持つ強力な統合開発環境で、MS-DOSやWindowsアプリケーション開発に必要なすべてのツールがすでに組み込まれています。さらにオーバーラップウインドウやマウス、そしてWindowsのプログラミングリファレンスを含んだオンラインヘルプをサポート。内蔵のマルチファイルエディタは、アンドゥ(実行の取消)／リドゥ(再実行)やマクロ機能を装えています。

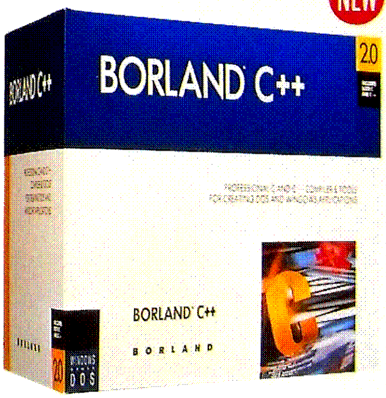
■動作環境：PC-9801シリーズ(ハイレゾモード対応、LTは除く)／メモリ：640KB以上／MS-DOS：Ver 3.0以上／MS-Windows 3.0以上／マウス：バスマウス／要ハードディスク

★お知らせ★

10月1日より、TURBO C++1.0を新価格にて販売いたします。——新価格 **¥20,000**(税別)
※スチューデント・パックの価格も変更になります。詳しくは、お問合せください。

■価格……………… **¥98,000**(税別)

NEW



（答） BORLAND® C++

※All Borland Products are trademarks or registered trademarks of Borland International, Inc ©Borland
※Microsoft、Windowsは米国マイクロソフト社の登録商標です。その他、商品名は一般に各社の商標です
●製品に関するお問い合せは、総販売元㈱マイクロソフトウェアアソシエイツ TEL 03-3486-1411までお願いします。

株式会社ボーランドジャパン
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400代

■新登場・Turbo Vision
もはや、アプリケーションを書く際にユーザインターフェース等の細かい部分で苦労する必要はありません。新しいTurbo Pascal 6.0に付属する、オブジェクト指向のユーザインターフェース・クラスライブラリTurbo Visionを使えば、プロ開発者レベルのすばらしいユーザインターフェースを持つアプリケーションを、驚くほど簡単に記述することができます。これによって、位置・サイズを自由に変更可能なオーバーラップウィンドゥやプルダウンメニュー、ダイアログボックス、ボタン等を駆使したマウス操作のイベントドリブン・アプリケーションを短時間で作成できます。さらに電卓、カレンダー、エディタ、時計、ディレクトリブラウザ、フォームといったアプリケーションをサンプルとしてソースコード形式で提供。これらを部品としてアプリケーションに組み込むことができます。事実、Turbo Pascal 6.0の新・統合開発環境もTurbo Visionを使って開発されています。つまり、操作性では定評のある統合開発環境と同レベルのアプリケーションが簡単に作成できる訳です。

■大きく改善された統合開発環境
新しい統合環境は、オーバーラップマルチウィンドゥをサポートし、マウス操作も可能です。エディタは、1MBまでのファイルを同時に複数オープンできます。また、内蔵デバッガでは条件ブレークポイント機能やレジスタウィンドウも追加されています。

■組み込みアセンブラ
従来のインラインアセンブル機能に加え組み込みアセンブラが使用できます。ソースプログラムにニーモニックを直接記述できるため、PASCALプログラム中でも簡単にアセンブラ言語を利用できます。

■ハイレゾルーションモードに対応
統合環境のハイレゾ31行モードでの動作、BGIやTurbo Visionでのハイレゾモードをサポート。また、H98シリーズの拡張アトリビュートにも対応しています。

■Turbo Pascal 6.0 Professional
Professional版には、タフなバグを始末するTurbo Debugger 2.01、プログラムのボトルネックを取り除くTurbo Profiler 1.0、MASMコンパチブルな高速アセンブラTurbo Assembler 2.0に加えて、プロテクトメモリを利用して巨大アプリケーションがコンパイルできるコマンドラインコンパイラが付属します。

■Turbo Pascal 6.0の動作環境
機種：PC-9801シリーズ、PC-286/386シリーズ（ハイレゾモード対応、LTは除く）／メモリ：640KB以上／MS-DOS：バージョン2.11以上／マウス：バスマウス
■価格：Turbo Pascal 6.0………20,000円(税別)
Turbo Pascal 6.0 Professional…39,800円(税別)

TURBO PASCAL® 6.0

※All Borland Products are trademaks or registered trademarks of Borland International, Inc. ©Borland International, Inc.
※その他、商品名は一般に各社の商標です。
●製品に関するお問い合せは、総販売元㈱マイクロソフトウェア アソシエイツ TEL.03-3486-1411までお願いします。

株式会社ボーランドジャパン
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400代

〈資料請求番号P02〉

32ビットパフォーマンスをDOSに与える！

DOS EXTENDER

- DOSのメモリ空間を、もっと大きく使いたい。
- 汎用機やEWSのプログラムを、PCに移植したい。
- 32bit PC本来の実行速度を、最大限に発揮させたい。

MS-DOSのアプリケーション開発におけるこのような要求に、DOS/4Gが応えます。DOS/4Gは、MS-DOSの640KBバリアを一気に解消し、アプリケーション・プログラムに最大4GB（ギガ・バイト）までの広大なメモリ空間を提供します。また、従来のMS-DOSでは、ハードウェア環境が32bitCPUであっても、16bit単位での動作しかされていませんでした。DOS/4Gは、CPUパフォーマンスをフルに引き出す32bitネィティブ・モード・プログラムを、DOS環境で実行可能にする、32bit専用DOSエクステンダーです。

■互換性を重視したDOSエクステンダ

DOS/4Gは、従来のDOSプログラムとの開発面での互換性を重視して作成された、32ビットDOSエクステンダです。開発プログラムが、32ビット・プロテクトモード・プログラムであることをほとんど気にすることなく、リアル・モード・メモリやビデオ・メモリへのアクセスや、MS-DOSの標準BIOSコールが行えます。

■DPMI（DOS PROTECTED MODE INTERFACE）規格にも完全準拠

DOS/4Gは、VCPI規格や、XMS、DPMIの全てのインターフェース規格に完全準拠しています。DOS/4Gアプリケーションは、XMS準拠のHIGHMEM.SYSや、VCPIに準拠したDesqView、QEMM.SYSや、MelwareなどのVCPIホスト下で、また、MS Windows 3.0/3.1を始めとする、あらゆるDPMIホスト下で動作及び他のプログラムとの共存が可能です。

■業界標準のDOSエクステンダ

16ビットDOSエクステンダである「DOS/16M」は、米国ではLOTUS、IBM、Ashton Tate、DEC、AutoDesk、Informixなど、300社をこえる大手ソフトウエア開発会社の主要製品に用いられ、数百万コピーのアプリケーションに組み込まれて出荷されています。国内でも、すでに200社を越えるソフトウエア開発会社に導入され、多くのソフトに組み込まれてすでに出荷されております。DOS/4Gは、この「DOS/16M」を基本に構築された、32ビット専用DOSエクステンダであり、パフォーマンスの高い32ビットアプリケーション開発のためのアップグレード・パスとなっています。

【対応機種】NEC PC-9801、東芝 J-3100＊、IBM PS/55＊、各社 AX仕様機種＊（JEGA.VGA AX)、各社DOS/V採用機種、IBM PC/AT,PS/2、（但し、それぞれ80386以上搭載機）
＊印＝デバッガは英語モードのみ、その他の開発・実行は日・英両モードで可能。

【動作環境】
MS-DOS Ver3.1以降
MS-Windows 3.0以降

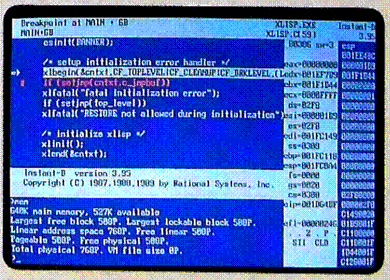
【対応言語】
WATCOM C/386
High C/386
Microsoftアセンブラ

	DOS/4G	DOS/4G用VMM
シングル・ライセンス	¥880,000	¥580,000
コーポレート・ライセンス	¥2,640,000	¥1,740,000
再販ライセンス	商用パッケージの再販のみ別途協議	

DOS/4G プロテクトモード・ユーティリティの概要

■高機能ソース・レベル・デバッガD32を装備

32ビットアプリケーションEXEファイルのデバッグが、アセンブラ・レベルとC言語のソースレベルの両方で行える、ウインドウタイプ高機能デバッガです。DOS/4Gで作成された32ビット・ネイティブ・プログラムが、従来のDOSプログラムをCodeView等でデバッグするのと、同様な感覚でデバックできます。



■32ビット専用リンカGLU32を装備

マイクロソフトLINKとコンパチブルなスクリプト・ファイルや、コンパチブルなオプションが使用可能な、32ビット・アプリケーション専用リンカです。

■バーチャル・メモリ・マネージャ VMN（別売オプション）

DOS/4Gのオプション製品である仮想記憶マネージャ「VMM」は、パソコンに実装されている以上のメモリ空間を使用するアプリケーションを実行可能にします。VMMは、80386/486ハードウエアのページング機能を活用し、コードとデータを自動的にディスクへスワップさせます。エンド・ユーザーは、RAMをあとから追加するだけで、実行速度を向上させることができます。VMMのAPIは、業界標準インターフェース規格DPMI 1.0の仮想記憶APIのスーパーセットであり、上位互換性があります。

Cプログラムに32ビットパワーを与える！

WATCOM C/386は、80386／486プロセッサが持つ32ビット・ネイティブモードのハイパフォマンス・オブジェクトコードを出力する高性能なCコンパイラです。コンパイラ自身が32ビット・ネイティブ・モードで動作するため、従来より遥かに大きなソースファイルが高速にコンパイル可能です。既存の16ビットDOS開発環境を、32ビットに移行することによって、DOSの640Kバイトメモリ制限を遥かに越えた、最大4G（ギガ）バイトまでのメモリを使用するパフォーマンスの高いアプリケーションがパソコン上で実現出来るようになります。しかも、効率の良い32ビットのリニア・アドレッシングでの大量データの高速アクセスがプロセッサ本来の能力を引き出して実現されるため、アプリケーション・プログラムの実行速度を大幅に向上します。

■80386サポート

WATCOM C/386は、ソースコードの変更や特別なコンパイラ・オプションを指定することなく、大きなデータエリアをアクセスできる32ビット386ネイティブモード・コードを生成します。386固有命令や、洗練された32ビットリニア・アドレッシングにより、WATCOM C/386は80386プロセッサの利点を効率よく最大限に引き出します。8086／286の64Kバイト毎のセグメントの境界や、DOSの640Kバイトの制限は、もはや存在しません。さらに、80486プロセッサも同様にサポートします。

■プロテクトモード・コンパイラ

WATCOM C/386パッケージには、リアルモードで動作するコンパイラとネイティブ・プロテクトモードで動作するプロテクトモード・コンパイラの2種類のコンパイラが含まれています。プロテクトモード・コンパイラを使用することによって、従来のリアルモード動作のコンパイラで使用していた640Kバイトメモリより遥かに大きなメモリを使用してコンパイルが行えるため、より大きなソースプログラムがオン・メモリで高速に処理できるようになります。この、プロテクトモード・コンパイラは、80386／486プロセッサ用DOSエクステンダ「DOS/4G」を使用して作成されています。

■ポータビリティ

既存アプリケーションの386アーキテクチャへの移行方法は、既存のソース・コードをWATCOM C/386で再コンパイルし、4Gバイトまでアクセス出来るように、WATCOM C/386ライブラリやDOSエクステンダ・モジュールとリンクするだけです。

■ANSI標準規格を完全にサポート

ANSI規格との互換性を調べるPlumHallバリデーション・スート・テストを完全に合格し、100%ANSI規格を満たしたCコンパイラであることが証明されています。

■強力なオプティマイザ

先進的なレジスタ割り当てと、命令の使用とともにフロー解析を行うことによって、生成されたコードをさらに最適化します。これによって、WATCOM社でテストした他の全ての80386 Cコンパイラの中で、最も速く、小さいコードを生成します。

■80387数値演算コプロセッサ・サポート

浮動小数点演算エミュレーションか、ネイティブ80387命令かを選択することができます。また、アプリケーション実行時にコプロセッサの有無によってエミュレート・ライブラリとネイティブ80387命令を判別実行出来るように、両方のコードをリンクすることも可能です。

■Microsoft Cとの互換性

Microsoft Cとのライブラリ及びソースレベルでの互換性によって、WATCOM C/386は、16ビットDOSアプリケーションを32ビットネイティブ・プロテクトモードに移植する理想的な言語処理系となっています。

■リエントラントな生成コード

WATCOM C/386によって生成されるコードは、リエントラントな構造を持つため、複数のスレッドの実行が同時にアクティブになる、割り込み処理等のアプリケーションが作成可能です。

■コンパイル管理ユーティリティ WMAKEを標準装備

■オブジェクト・コードの逆アセンブル・ユーティリティ WDISASMを標準装備

■32ビット・DOSエクステンダ「DOS/4G」、「386|DOS EXTENDER」の両方をサポート

【対応種類】PC-9801シリーズ、PC/AT互換機（J3100シリーズ、各社AX仕様機、PS/55シリーズ）（但し、80386又は80486プロセッサ搭載機に限る）
【対応DOSエクステンダ】DOS/4G、386|DOS EXTENDER
【動作環境】MS-DOS V3.1以上
【必要メモリ】2.6Mバイト以上

シングル・ライセンス	¥198,000
コーポレート・ライセンス	¥594,000

お問い合わせは…㈱LIFEBOAT 営業1課（tel.03-3293-4714）までご連絡ください。

●Copyright Notice —— ＊DOS/4G, DOS/16Mは、米国Rational Systems Inc.の登録商標です。＊WATCOM Cは、加国WATCOM Systems Inc.の登録商標です。＊MS-DOS, MSはMicrosoft Corp.の登録商標です。＊その他、プログラム名、システム名、CPU名は一般に各メーカーの登録商標です。

株式会社 LIFEBOAT
〒101 東京都千代田区神田錦町3-6
PHONE:03-3293-4711代 FAX:03-3293-4710

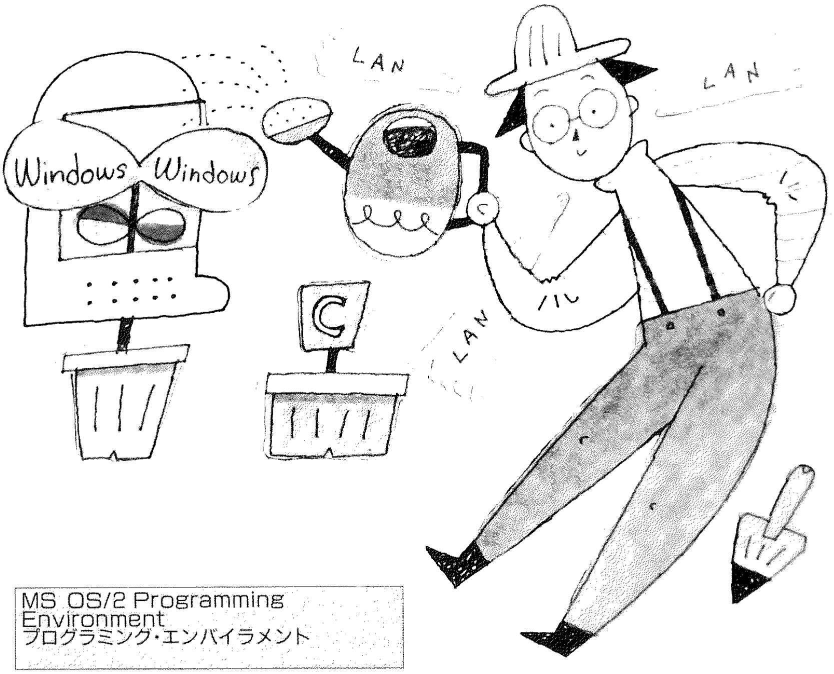
＜資料請求番号001＞

次代のアドバンテージ、最新の技術をあなたに!

Microsoft® University

独学より大学。

マルチメディアのプラットフォームMS-Windows、高信頼性のMS OS/2。本格的なマルチタスクの実現、グラフィカルなユーザ・インタフェース(GUI)、LANマネージャによる高度なネットワーク処理など、技術者の創造力を刺激する環境を提供します。マイクロソフト・ユニバーシティは、その登竜門。ウインドウ環境、デバイスドライバの開発、ネットワーク処理など、アプリケーション・ソフト開発に欠かせない、ノウハウをレクチャします。入門から専門まで、充実したコースラインアップ。短時間で即開発に結びつく新しいコンピューティングスタイルを身につけて下さい。



Microsoft Windows Programming Environment
ウインドウズ・プログラミング・エンバイラメント

■期間：5日間

■MS Windows Ver3.0のアプリケーションを作成する方のためのコースです。Windows プログラミングのポイントを重点的に解説、実習中心のコースです。

Microsoft Windows Applications Programming
ウインドウズ・アプリケーションズ・プログラミング

■期間：5日間

■MS Windows Ver3.0対応のアプリケーションを開発するプログラマや設計者のためのコースです。ユーザインタフェースやアプリケーションなど設計ポイントがマスタできます。

Microsoft LAN Manager Programming Environment
LANマネージャ・プログラミング・エンバイラメント

■期間：4日間

■マイクロソフトLANマネージャ(LM)用アプリケーションを開発する設計者のためのコースです。LMのアーキテクチャと特徴について解説します。

Microsoft LAN Manager Applications Programming
LANマネージャ・アプリケーションズ・プログラミング

■期間：5日間

■マイクロソフトLANマネージャ(LM)上で動作するアプリケーションの作成や移植するプログラマのためのコースです。

MS OS/2 Programming Environment
プログラミング・エンバイラメント

■期間：4日間

■MS OS/2のアプリケーションを作成するために必要となる基本的技術を修得するコースです。本コースは、OS/2に取り組む上での基本コースとなります。

MS OS/2 Presentation Manager Programming Environment
プレゼンテーション・マネージャ・プログラミング・エンバイラメント

■期間：5日間

■MS OS/2プレゼンテーション・マネージャのアプリケーションを作成するための入門コースです。作成に必要となる基本事項を説明いたします。

MS OS/2 Applications Programming
アプリケーションズ・プログラミング

■期間：5日間

■MS OS/2のAPIを使用し、マルチタスク、メモリ管理、プロセス間通信、ダイナミックリンクなどの機能を有効に活用する、実習を中心としたコースです。

C言語入門

■期間：2日間

■これからC言語を始めようとする方を対象に、C言語の基本的な文法からわかりやすく解説します。統合環境を持つQuick Cを使って理解を深めます。

MS OS/2 Presentation Manager Applications Programming
プレゼンテーション・マネージャ・アプリケーションズ・プログラミング

■期間：5日間

■MS OS/2 PMのアプリケーションを作成するためのステップアップコースです。経験者の方を対象に、より詳しく説明します。

MS OS/2 Device Drivers
デバイス・ドライバ

■期間：4日間

■MS OS/2のデバイス・ドライバの機能、構造を修得し、実際にデバイス・ドライバを作成するコースです。デバイスヘルパールーチンなども解説します。

C言語応用

■期間：3日間

■関数、ポインタ、構造体についてプログラムを作成しながら詳しく説明します。セミナは、最新のMicrosoft C Ver 6.0を使用して実施いたします。

●お問い合せは
富士ソフトウエア株式会社 MU教育センター
TEL 03-3452-7722(直通) FAX.03-3798-0599
日本電気株式会社殿の御協力によりPC-9800シリーズ対応コースも好評開催中!!

Microsoft®
マイクロソフト株式会社
〒160 東京都新宿区西新宿7-5-25 Kビルディング

FSI 富士ソフトウエア株式会社
MU教育センター
〒108 東京都港区三田3-4-3 三田第一長岡ビル

〈資料請求番号002〉

DOS-Extender対応

最適化 FORTRANコンパイラ

NDP FORTRAN 386

NDP FORTRAN 386は80386のプロテクトモードに対応したコンパイラで、386|DOS-Extenderを使用し、実装されているメモリを単一レベルで確保。メモリの許すかぎりの巨大配列を定義した巨大プログラムを実行できます。もう640Kバイトの壁に悩まされることはありません。

FORTRAN77のフルセットコンパイラであり、ANSI X-3.9-1987とDOD MIL STD 1753、さらにUINX 4.2BSDの拡張機能を完全に満たしているため、ミニコンやワークステーション、メインフレームのプログラムの移植が容易です。またNDP FORTRAN386は、きわめて高速なオブジェクトを生成します。特に配列変数のアクセスはDOS上のコンパイラの数倍から10数倍のスピードのオブジェクトを出力します。インラインコードの生成も可能です。

- ●シンボル名として31文字まで使用できます。$の使用も可能です。
- ●INCLUDEのネストが可能です。
- ●MIL STD-1753のビット操作を全てサポートしています。IOR、IAND、IEOR、NOT、ISHFT、ISHFTC、ISHFTL、ISHFTR、BTEST、IBSET、IBCLR、MVBITS。
- ●ベッセル関数、ガンマ関数、誤差関数をサポートしています。
- ●ZとO編集が入出力に使用できます。
- ●FORTRAN 66のDOループをオプションスイッチにより選択できます。
- ●カンマを使用したフリーフォーマット入力をサポートしています。
- ●$を入出力の改行文字として利用できます。
- ●カラム1にX, Dを記入することにより、条件付きコンパイルができます。
- ●COMMONとEQUIVALENCEにおいて文字と数字を混在できます。
- ●コマンドラインに指定したGETARG、IARGC、GETENVを読み込む機能をサポートしています。
- ●次の型が使用できます。
  REAL＊4、REAL＊8、INTEGER＊1、INTEGER＊2、INTEGER＊4、LOGICAL＊1、LOGICAL＊2、LOGICAL＊4、COMPLEX＊8、COMPLEX＊16。
  デフォルトのLOGICAL, INTEGERは4ですが、それを2に変更することもできます。

定価 ¥148,000

32ビット開発ツールキット

386|TOOLBOX

386|TOOLBOXは80386/486CPUのプロテクトモード専用のプログラム開発キットです。32ビットプロテクトモードを管理する386|DOS-Extenderのほか、32ビット対応のアセンブラ、リンカ、ライブラリアン、デバッガを一つにまとめたパッケージで、386/486の全ての命令セットを使用したアプリケーションを開発可能。ユーザーメモリは640Kバイトだけでなく、実装されている全てのメモリを利用可能です。また386|TOOLBOXはMS-DOSの環境を利用しているので、従来のエディターなどがそのまま利用可能です。開発した製品を販売する際は、ランタイムライセンス(¥600,000)が必要となりますが、開発のみの場合にはまったく必要ありません。

386|DOS-Extender

386|DOS-Extenderは、MS-DOSの環境下で32ビットCPUのプロテクトモード環境を実現し、MS-DOSとプロテクトモード間のレイヤとして機能します。これにより、ユーザーメモリは640Kバイトだけでなく、実装されている全てのメモリを利用可能にします。
MS-DOSのシステムコールのほかに、リアルモードの手続きを呼び出したり、リアルモード、プロテクトモードの割り込みを管理するための拡張ファンクションコールを独自に所有しています。

■主な拡張ファンクションコールリスト

2502H	プロテクトモードの割り込みベクターの取得
2503H	リアルモードの割り込みベクターの取得
2504H	プロテクトモードの割り込みベクターのセット
2505H	リアルモードの割り込みベクターのセット
2506H	常にプロテクトモード下で動作する割り込みのセット
2507H	プロテクト/リアルモードの割り込みベクターのセット
250CH	ハードウェア割り込みベクターの取得
250DH	リアルモードとリンク情報の取得
250EH	リアルモード手続きの呼び出し
250FH	プロテクトモードアドレスからMS-DOSのアドレスへの変換
2510H	レジスタ指定付きのリアルモード手続きの呼び出し
2511H	レジスタ指定付きのリアルモード割り込みの実行
2516H	LDTが占めるメモリの解放
2517H	DOSのデータバッファに関する情報の取得

386|ASM, 386|ASMP

- ●ソースプログラム形成はマイクロソフト社のMASM Ver5とほぼ完全な上位互換性があります。
- ●8086から80486までのプロテクトモードを含む全命令をアセンブル可能です。
- ●32ビット対応の疑似命令セットをサポートしています。
- ●リアルモードアセンブラ(386|ASM)と386/486のプロテクトモードアセンブラ(386|ASMP)の2つを用意。32ビット対応コンパイラのNDP-FORTRANやHIGH-Cとリンク可能です。
- ●オブジェクトモジュールフォーマットはIntel社の8086用OMF-86、およびOMF-86と上位互換のある簡易OMF-386を出力します。

386|LINK

386|LINKはIntel社の386/486CPU用のリンカで、Intel OMF-86および簡易OMF-386ファイルをリンク可能です。また386|ASMやHIGH-C、NDP-FORTRANのオブジェクトモジュールをリンクするほかに、MS-CやMASM等のOMF-86互換もオブジェクトモジュールのリンク可能です。

定価 ¥148,000

	MS-DOS	コプロセッサ	プロテクトモードメモリ	ハードディスク	PC-9801/H98	PC/AT互換	FMR
386|TOOLBOX	要	対応	1Mbyte以上	要	○	○	○
386|VMM	要	対応	1Mbyte以上	要	○	○	○
386|DEBUG	要	対応	1Mbyte以上	要	○	○	○
NDP FORTRAN386	要	必須	2Mbyte以上	要	○	○	
NDP C386	要	必須	2Mbyte以上	要	○	○	
HIGH C386	要	対応	2Mbyte以上	要	○	○	○

※上記の製品は、すべて32ビット機のみに対応します。また、これらの製品は、使用する機種別(NEC、IBM、富士通)に購入していただく必要があります。

お買い得パックセット

386|TOOLBOX＋386|DEBUG＋386|VMM	¥248,200
386|TOOLBOX＋386|DEBUG	¥175,100
386|TOOLBOX＋386|VMM	¥198,900
386|DEBUG＋386|VMM	¥129,600

※すべての価格に消費税は含まれていません。

テックソフト&サービス株式会社
コンピュータ事業部
〒273 千葉県船橋市本町7-11-5 レランドセンタービル5F
TEL.(0474)23-1322 FAX.(0474)23-0202
ユーザーサービス直通
TEL.(0474)22-9955(月～金の10:00～12:00、13:00～16:00)

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集

286/386メモリ活用プログラミング

MS-DOSからプロテクトメモリを利用するために

EMS，XMSを中心に…
MS-DOSメモリ管理方式の歩み 加野島英渡 2

仮想記憶モードをもった高機能CPUの概要
80286，80386のアーキテクチャ 秋山登 13

386によるプロテクトモードプログラミングの実際
プログラム実行環境「P386」の作成 秋山登 28

プロテクトメモリ管理規約の概要
MS-DOSのメモリ拡張とVCPI，DPMI 菱博善 45

80286，80386のプロテクトモードを使う
DOSエクステンダ活用プログラミング 森 誠 52

プログラミングはどう変わるのか
フリーエリアが広がったMS-DOS 5.0 インタビュー●マイクロソフト(株) 69

IC千夜一夜 第4回
初期のメモリ素子 ―遅延線メモリとコアメモリー 虎岩登 78

MS-DOSツール・ラボ 第7回
FAST基本ライブラリ“コア”FORTHコンパイラの作成③FAST 加野島英渡 82

実践C塾 基礎テクニック篇 第4回
辞書順ソート 大野元久 94

98活用アセンブラ技法／番外編 第2回
書式およびデバッグのコツ 星野操 104

付録
5” 2HD フロッピーディスク

綴じ込み付録
EMSファンクションリクエスト(INT 67H)一覧

表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

パソコン博物館　第10回
クリーンコンピュータ／シャープ MZ-80C　井戸禎光　74
プログラマの道具学　第10回
ジャンクフード　祐安重夫　76
情報処理技術者試験 第2種合格／トレーニング
第7回　プログラム流れ図　望月照夫・沖山貴裕　113
Book Review「SGML 入門」　中村正三郎　118
本のアンテナ　古田島義和
佐々木哲のニューヨークシステム通信　120
西川和久の希望的観測　122
コンピュータの法律　熊谷信太郎　124
アキバ短信
プログラマの仕事場　第7回　株式会社システムハウスミルキーウェイ　126

INFORMATION PAGE　129
NEWS & NEW PRODUCTS　130
Programmer's Program　Mortice Kern Systems ツールキット　134
オンラインソフト採集帳10「DENEC」　136
HEALTHY? UNHEALTHY?
ドラゴン・ポスト　138
付録ディスクのご案内　140
バーチャルスペースガイド　141
読者プレゼント
クイズ・ランダムスクリーン　142
オンラインサービス（SE-NET）のお知らせ
バックナンバーのご案内　143

広告索引　資料請求番号

(株)ボーランドジャパン……P02
(株)ライフボート……001
富士ソフトウェア(株)……002
テックソフト＆サービス(株)……003
T・ZONE……004
(株)ビレッジセンター……005
日本システム(株)……P03
三菱電機(株)……P04

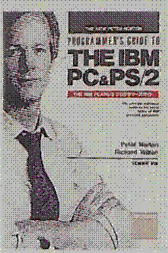
人材募集広告

(株)エルテックス
バーテックシステム(株)
ロジック・アーツ(株)
(掲載順)

1991 OCTOBER No.10

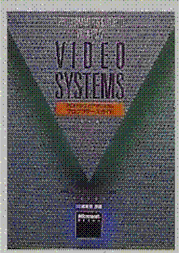
翔泳社が贈るプログラマ必携書

IBM PC & PS/2 プログラミング 3部作



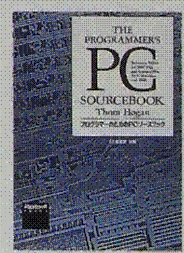
あのピーター・ノートンのベストセラー，IBM プログラマ必携のスタンダードガイダンス．

IBM PC & PS/2
プログラマーズガイド
● Peter Norton & Richard Wilton著
●定価　5,200円



IBM PC ファミリーのビデオシステム完全ガイド．

PC & PS/2 ビデオシステム
プログラマーズガイド
● Richard Wilton著
●定価　6,000円



IBM PC & PS/2 プログラマ必携の技術資料完訳．

プログラマーのための
PC ソースブック
● Thom Hogan著
●定価　5,200円

IBM Authorized Dealer

今月の特選お買い得品！

T·ZONE

80486/33MHz made in USA
先着20台限り 超特価¥498,000

80486/33MHz
RAM 4MB(max 32MB)
64KBキャッシュ(max 256KB)
3.5"1.44MB×1/5"1.2MB×1
CONNER 120MB HDD
スーパーVGAアダプタ512KB
2シリアル、1パラレル
拡張スロット×8
ドライブベイ×6
ミニタワー
アルプス製101キーボード

80386/33MHz ノート made in TAIWAN
限定20台 超特価¥498,000

80386/33MHz
RAM 2MB(max 16MB)
32KBキャッシュ
3.5"1.44MB×1
40MB HDD
32階調VGAバックライト液晶
外部VGA出力
1シリアル、1パラレル
外部テンキー付属
外部101キーボード、外部FDDポート
ニッカドバッテリー(充電2時間、2時間使用)
280W×220D×54H 4kg

Mitac

3060D	101キーセット	106キーセット
RAM 3MB、HDD 40MB	¥318,000	¥333,000
RAM 3MB、HDD 120MB	¥368,000	¥383,000
RAM 5MB、HDD 120MB	¥385,000	¥400,000

共通仕様
80386SX/16MHz、3.5"1.44MB×1、5"1.2MB×1、スーパーVGA(512KB)、スーパーVGAモニター

MIMI-SCSI 超特価¥29,800
プリンタポートにつなぐだけでSCSIデバイスが使えるようになります。
87MB HDDセット 超特価¥138,000

マウス シリアル 超特価¥6,800〜 PS/2 超特価¥8,000〜

PS/55 NOTE用 2MBメモリー 超特価¥28,000〜
4MB 超特価¥48,000〜

Microsoft BALLPOINT MOUSE 超特価¥28,000

VGA信号をビデオに変換するコンバーター 超特価¥98,000
320×200〜640×480の信号をS-VHS、ビデオに変換して、テレビやビデオに接続できます。

SCSI I/F 超特価¥9,800〜 SCSI I/F, W/FDC 超特価¥15,000〜

WINDOWS 3.0
時代のトレンドWINDOWS 3.0がIBMではマルチウインドウズで複数のDOSアプリが同時実行可能です。

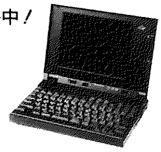
IBM	日本語WINDOWS 3.0	¥20,000
ADOBE TYPE MANAGER		¥20,000
AFTER DARK		¥9,800
BECKERTOOLS		¥16,000

MICROSOFT ENTERTAINMENT PACK ¥8,800
その他WINDOWS用アプリケーションが200種類以上あります。
WINDOWS 3.0対応のPDS(英語)もあります。

NOTE
ノートといえば、PS/55 note 大特価提供中！

IBM	PS/5523SJ1	~~¥248,000~~
IBM	PS/5523SJ4	~~¥368,000~~
NBCC	386NOTE	¥298,000
NBCC	386LAPTOP	¥266,000
NBCC	286NOTE	¥244,000
NBCC	286LAPTOP	¥220,000

PS/55 note



DOS/V
DOS/VといえばT·ZONEです。入門機から互換機まで！

IBM	PS/5530-SW4	大特価提供中！ ~~¥609,800~~
	標準で6MBのRAMと日本語WINDOWS 3.0が付属しています。勿論DOS/Vも付いています。	
IBM	PS/5510ZJ2	大特価提供中！ ~~¥214,000~~
IBM	PS/5510ZJ4	大特価提供中！ ~~¥308,000~~
IBM	PS/5510TJ4	大特価提供中！ ~~¥564,000~~
ALR	POWER FLEX 20CSX, RAM 5MB, 3.5" 85", 40MBHD	¥385,000
ALR	BUSINESS VEISA 486/33, RAM, 2MB, 3.5" 85", 120MBHD	~~¥1,250,000~~
東芝	DynabookでDOS/Vが動いています(2Fでデモ中)	

VGA
DOS/Vの必須アイテムはVGAアダプターです。豊富な在庫から選べます。

BASIC VGA 640×480 ¥19,800
SUPER-VGA 1024×768 16色 ¥38,000
SUPER-VGA 1024×768 256色 ¥44,800
Dyanbook用VGAアダプタ ¥49,800

VGAモニター
DOS/V対応やWINDOWS用として高解像大画面のモニターがほしいあなた、何を選びますか？

VORTEC CM-1413 14" 0.28mm 15.0-38kHz 1024×768 ¥68,000
デジタル・アナログ両用

三菱	XC-98V3	14" 0.28mm 98/VGA	640×460	¥78,000
ACER	7015	14" 0.31mm 15.0-36kHz	800×600	¥85,000
日立	CM1483MU	14" 0.28mm 30-40kHz	1024×768	¥89,000
NEC	MULTISYNC II	14" 0.31mm 15.5-35kHz	800×560	¥95,000
NEC	MULTISYNC 3D	14" 0.28mm 15.7-38kHz	1024×768	¥180,000
NEC	MULTISYNC 4D	16" 0.28mm 30.0-57kHz	1024×768	¥280,000
IBM	8515	14" 0.28mm VGA, 8514/A, XGA		~~¥160,000~~

お知らせ

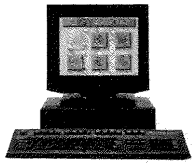
DOS/Vプログラムコンテスト開催。〜9月末
DOS V用のユーティリティーやFEP、表計算ソフト、エディターなど、どんなジャンルのソフトでもOKです。
PDS、フリーウェア、シェアウェアをどしどし作って下さい。
DOS V用であれば何でもOK！
特選1名様に「テラドライブ」をプレゼント……他、お申し込みの方全員にもれなく商品をさしあげます。ふるってご応募下さい。

GAME
時代はゲーム!? TERADRIVEや新作のソフトが山盛りでご来店をお待ちいたしております。

SEGA TERADRIVE MODEL1	~~¥148,000~~	大特価提供中！
SEGA TERADRIVE MODEL2	~~¥188,000~~	大特価提供中！
SEGA TERADRIVE MODEL3	~~¥248,000~~	大特価提供中！
SEGA TERADRIVE 専用モニター	~~¥89,800~~	大特価提供中！
SEGA TERADRIVE マウス	~~¥6,800~~	大特価提供中！

WALT DISNEYの永遠のアイドル、ミッキーマウスがあなたにしゃべりかけます。

MICKEY'S ABC	¥10,800
MICKEY'S 123	¥10,800
MICKEY'S ABC & SOUND SOURCE	¥13,800
SOUND SOURCE(簡単のローコスト音源スピーカーです)	¥6,800
ADLIB FOR ATバス	¥30,000
ADLIB FOR MCAバス	¥40,000
SOUND BLASTER FOR ATバス	¥48,000
SOUND BLASTER FOR MCAバス	¥68,000
JOYSTICK	¥4,800
ゲームカード(PC/AT)	¥9,800
ゲームカード(MCA)	¥15,000



仕事も遊びも ちょっと、こだわる。そこで…

世界のソフトウェアが走る。

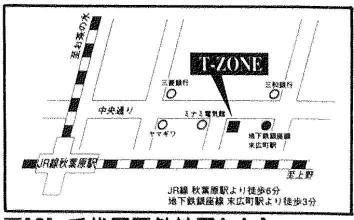
DOS/V OADG

PS PLAZA のサポート拠点です！

T·ZONEは今年1月19日からDOS/Vのサポート拠点として「PS PLAZA」をオープンしております。

特徴その1 日本IBMのSEが親切にどんなご相談にもお答えします。
特徴その2 いろいろな互換機上でDOS/Vが動くマルチプラットフォーム環境が揃っています。
特徴その3 PS PLAZAの直通番号があります。(03)3257-2654
特徴その4 外人さん大歓迎！英語、韓国語、ポルトガル語、イタリー語、スペイン語、バングラ語などのマルチリンガルOKです。
特徴その5 AT環境のボード、ソフトが日本一！揃っています。
特徴その6 DOS/V、WINDOWS、LAN、マルチメディアに関する、ホットな最新情報が得られます。

T·ZONE
IBMゾーン
☎03-3257-2652
3F ハードウェア
4F ソフトウェア



〒101 千代田区外神田4-4-1
☎03-3257-2652 FAX 03-3257-2661

T·ZONEスタッフ募集 IBM、Apple、無線のスタッフを募集中です。アルバイトも募集中！ ☎03-3257-2690

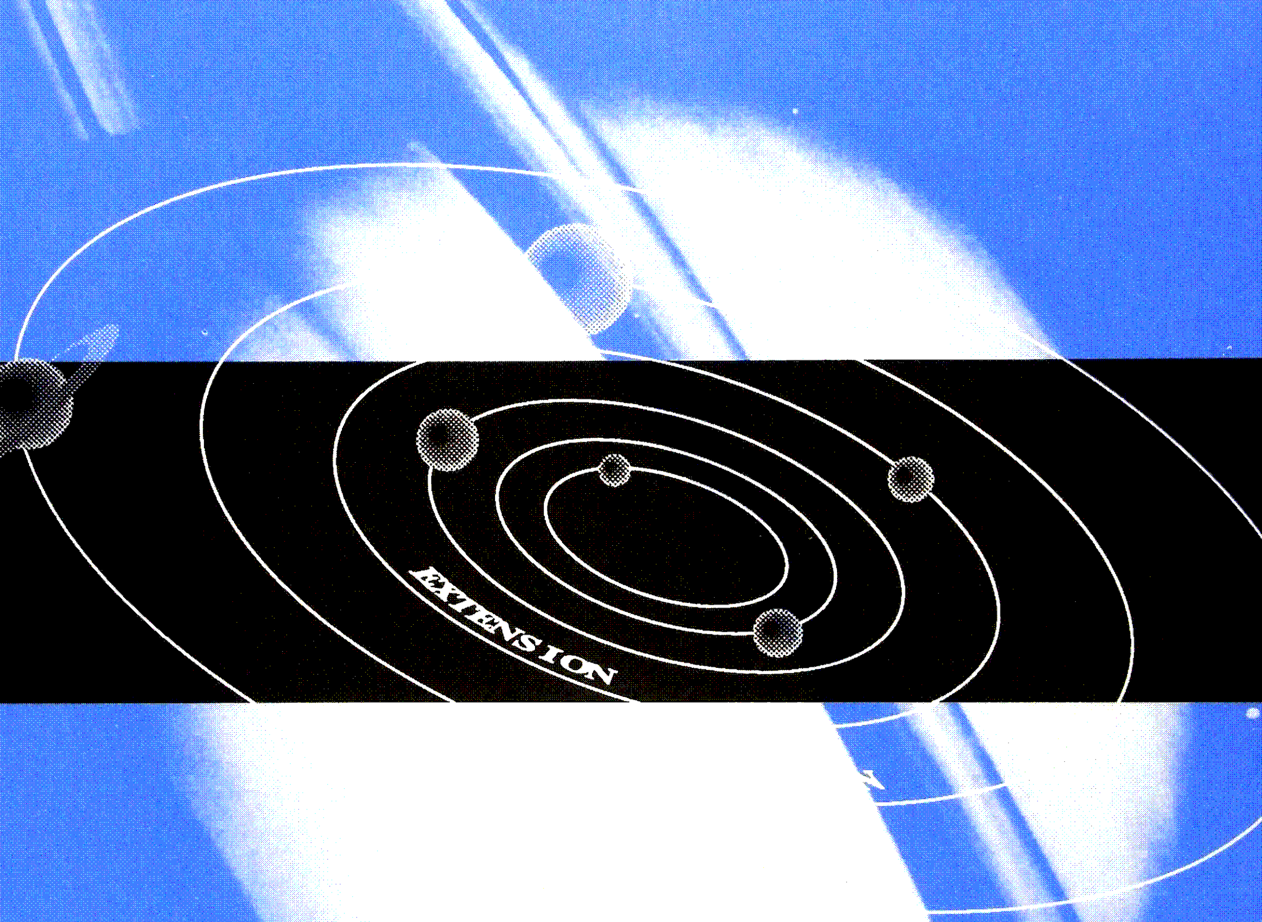
☆T·ZONEの7Fは専門書籍(国内・外)・データーブックの専門フロアです。大量在庫中！ ほしい本は何でも揃います。

〈資料請求番号004〉

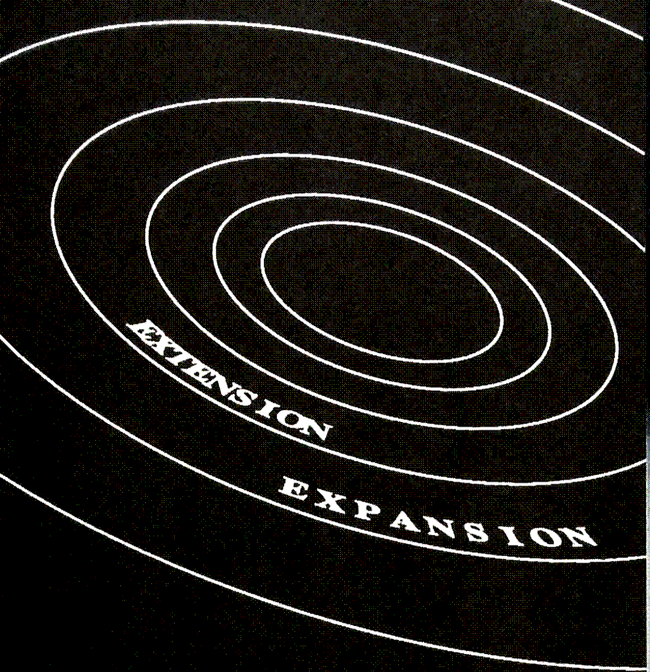
特集

286/386 メモリ活用プログラミング

MS-DOS からプロテクトメモリを利用するために



その昔，CP/M から MS-DOS へと移ってきたプログラマの多くが640KB という“広大な”メモリ空間に驚きを覚え，この広いメモリを使いきるプログラムなど書けるわけがないと思ったものである．しかし，それからわずか数年で，この640KB のメモリ空間は“壁”となってプログラマの前に立ちはだかることになる．以後，MS-DOS 上でプログラミングを行うプログラマ達は，この640 KB のバリヤーを打ち破るべく数多くのテクニックを開発してきた．オーバーレイ，バンク式メモリ，EMS，そして進化した CPU である80286，80386の機能を利用したプロテクトモードの活用などである．本特集では，おもに80286，80386のアーキテクチャとその機能を利用して MS-DOS 上からプロテクトメモリを活用する方法を解説するとともに，DOS エクステンダの利用方法，VCPI，DPMI の意義およびユーザーメモリの拡張機能を備えた DOS 5.0について解説する．



特集

286/386メモリ活用プログラミング

EMS，XMSを中心に…

MS-DOS メモリ管理方式の歩み

加野島英渡

MS-DOSのメモリ不足を解決するために，これまでさまざまな方法が試みられてきた．640KBのメモリ中に必要に応じてプログラムを読み込んだり，切り放したりするオーバーレイ，メモリの一部分を窓のようにして切り替えるバンク方式やEMS，A20アドレス線を有効にすることでアクセス可能になるHMA，640KB～1MBにわたるROM領域の"空き"を利用するUMB，さらに，プロテクトモードをMS-DOSから利用するEMB．
ここでは，バンク方式からプロテクトモードの利用へといたるMS-DOSにおけるメモリ管理方式の変遷を眺めてみよう．

はじめに

冒頭から断言してしまうが，1985年は拡張メモリのひとつの出発点の年であった．世間では，「つくば科学万博」などというものをやっていた年だし，あの阪神タイガースが優勝したころの話だ．わずか6年前ではあるが，人によってはひと昔前という気もするかもしれない．

この年，日本のパソコン市場のひとつの原器ともいうべきPC-9801VMが発売された．128KBからスタートしたPC-9801シリーズのメモリは，このあと発売されたPC-9801VXに至って，とうとう640KBのフル実装となっていくのだが，それと歩調を合わせるかのように，高機能化していくアプリケーションが続出し，しだいにメモリ容量の限界が話題になってきた．そう，640KBという壁はもう目前だったのだ．何とかしてこの壁を打ち破り，もっと広い世界で思う存分プログラムを動かしたい……，「パソコン本体と別にお金を出してメモリというものを買い求める」という事実が，特殊なマニアだけのものでなくなってきたのは，まさにこの1985年あたりだといえよう．その中でまず脚光をあびたのが「バンク方式」と呼ばれるメモリの拡張方法だった．さらにこの年，今や時代の寵児となった感がある「EMS」がアメリカで産声をあげている．ただ，当時はまだマイクロソフト社も企画に加わっておらず，日本でも一般にはあまり関心を寄せられない，そんなスタートであった．

CP/Mの64KBの世界から比べると，「広大」なメモリ空間を獲得したかにみえたMS-DOSだが，高機能で巨大なアプリケーションは加速度的にさらなるメモリを要求しはじめ，640KBの向こうの世界をめざしてきた．本章では，MS-DOSがいかにして640KBの向こう側を獲得してきたかを，代表的なアプローチの解説を通して見ていきたいと思う．

バンク方式

最初に地位を獲得したのは「バンク方式」と呼ばれる拡張方法だった．前述したように，以前のパソコンはMS-DOSが直接管理できる最大容量分のメモリを最初から搭載していたわけではなかった．今では考えられないことだが，PC-9801シリーズの場合は「640KBにするためにオプションでメモリを購入する」ということになっていた．

バンク方式は，この「本体を買っただけではRAMが実装されていないメモリ空間」を利用して，640KBの最後の128KBに特殊なハードウェアの仕掛をした拡張方法だ．簡単にいうと，最後の128KBを「窓」のように考え，ハードウェアを操作することによって，その「窓」に別々のデータが現れるようにしたものといえる．たとえば，その「窓」は紙芝居の枠のようなもので，操作によって次々と違う絵を見ることができる原理になっているというイメージである（図1）．

この「バンク方式」でいちばん普及した用途は，RAMディスクとしてこ

コラム1 拡張メモリのスピード

PC-9801にメモリを増設する場合，本体後ろの「拡張スロット」に入るメモリボードを購入するのと，本体内部の「メモリ専用スロット」に入るメモリを購入するという2つの選択が考えられる．今のところ，価格が安い，機種を変えても確実に転用できるという点から拡張スロットに増設する人が多いようだが，Windows 3.0など本格的に大量のメモリを使用する環境が出てきて，そのスピードの差がクローズアップされてきているようだ．

「拡張スロット」は，たとえ本体が32ビット機でも，16ビット単位でデータを処理するうえに，さまざまな拡張ボードに対応するために，動作速度が低く押さえられている．一方の「メモリ専用スロット」は，本体のメインボードに直結するわけだから，本体の性能がよくなればなるほど，その速度は向上していく．32ビット機に拡張スロットのメモリと専用スロットのメモリを両方増設して電源を入れると，RAMチェックの数字の上がり方が極端に違うのが実感できる．

ただ，RAMボードの値段は下がってきたとはいえ，まだまだ専用スロットに入れるRAMは割り高だし，本体を買い替えたときにせっかく買ったRAMが使えないのでは，という不安も残る．1日も早く，32ビット機の性能を活かすRAMを安価に安心して買えるようになってほしいと願う今日この頃である．

図1
バンク方式の考え方

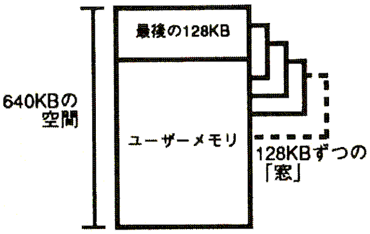
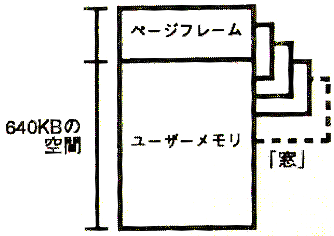


図2
EMSの考え方



の空間を利用するものであろう．ハードディスクが高嶺の花だった時代，RAMディスクのアクセススピードはめくるめくような速さだった．

だが，この「バンク方式」もいろいろな問題点をかかえていた．まず第一に，ハードウェアに特別な仕掛が必要なため，特定のRAMボードを購入しないかぎり利用できないという点．そして標準搭載メモリが640KBに達したとき，今度は反対に最初から付いているメモリを切りはなして使わなければならなくなったという点．何よりもスタンダードなものになるのを妨げたのは，この方式でメモリを拡張するための標準的な手法（プログラムインターフェース）が存在しなかったことだ．複数のプログラムが，それぞれ独自の方法でハードウェアを操作していれば，当然お互いに共存できるはずはなく，またアプリケーションレベルで，直接このハードウェアをアクセスするのはたいへん難しくなってくる．標準化されたプログラムインターフェース，これがメモリ拡張にもっとも必要であることをこの「バンク方式」は教えてくれた．

後に標準化を志向した「BMS」というプログラムインターフェースも提案されたが，この時点ではすでに後述する「EMS」が市民権を獲得しつつあり，EMSを凌ぐ勢力には至らなかったようだ．

EMS (Expanded Memory Specification)

EMSの背景

「EMS」は，Lotus社（Lotus1-2-3のメーカー），Intel社（ご存じ80x86のメーカー），Microsoft社（これもご存じMS-DOSのメーカー）という，パソコン界の中心的な3社が提唱したメモリ拡張方式だ．原理的には，先ほどのバンク方式と共通するものがあり，特定のアドレス空間を「窓」（EMSではこれを「ページフレーム」と呼ぶ）として，ここを通してより広いメモリ空間を「覗き見る」わけだ（図2）．

バンク方式と違うのは，前者がユーザーメモリの最後の128KBに仕掛を施したのに対し，EMSではユーザーメモリ空間の外に「窓」を設定できるということであろう．これによって，従来のメモリ空間に影響を与えることなくメモリの拡張が可能になっている．だが何よりもEMSが画期的だったのは，Lotus-Intel-Microsoftというパソコン界に大きな発言力をもつ3社がメモリ拡張方法としてその仕様を規定したということだろう．先ほど書いた「標準化されたプログラムインターフェース」として規格されたもの，つまりEMSは最初から「標準」となるべくして産み出されたものなのだ．

EMSの実現方法

EMSを理解するためにまず「ページ」という用語を覚えなくてはならない．EMSは，メモリ空間を16KBを1単位とする「ページ」で管理する．そして，アプリケーションから直接見ることのできる窓（ページフレーム）のことを「物理ページ」，その窓を通して切り替える拡張メモリ領域を「論理ページ」と呼ぶ．図3の場合，ページフ

メモリ管理用語集
IDT（Interrupt Descriptor Table）　プロテクトモードの割り込みベクタテーブル．

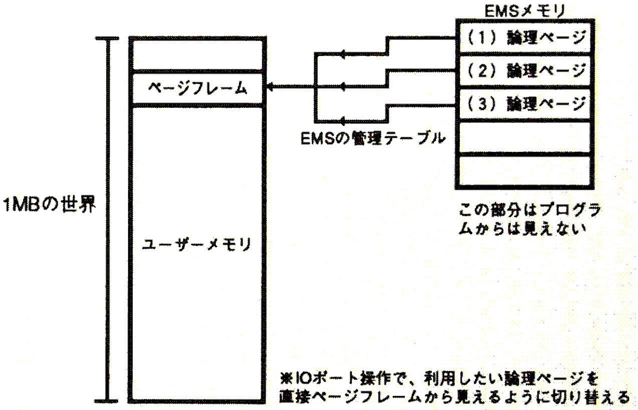
レームには３つの物理ページ（つまり16×３＝48KB）が確保されている．EMSとは，ページフレーム(物理ページ）を通して論理ページという単位に分割された広い拡張メモリの世界を覗き見ることと理解されたい．

ところで，ひとくちにEMSといっても，仕様が決められているのはそのプログラムインターフェースであり，そのインターフェースを実現するための方法は規定されていない．EMSを実現するには，一般に「EMSドライバ」と呼ばれるデバイスドライバを使用するが，その実現方法には特別なハードウェアを必要とする方式やソフトウェアのみで行う方式などが存在する．方式の違いは，それぞれの「EMSドライバ」が吸収し，アプリケーションがEMSを使用するときは，どの方式か？　などということは気にすることなく，標準化された手順で可能なことはいうまでもない．

現在，一般に使われている代表的な実現方法は，それぞれ「ハードウェアEMS」「ソフトウェアEMS」「仮想86モードEMS」と呼ばれている．以下，これらの動作原理を見ていくことにしよう．

図4

ハードウェアEMSの原理

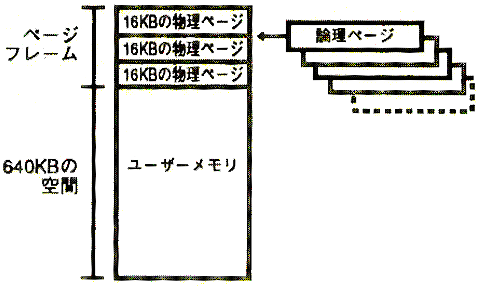


ハードウェアEMS

ハードウェアEMSは，アプリケーションの希望する「論理ページ」を，ハードウェアの操作によってページフレーム(物理ページ)に出現させるという，基本的にはバンク方式とたいへんよく似た方式だ．このため，バンク方式と同様に専用のハードウェア（ハードウェアEMS対応のメモリボード）が必要になる．また，このようなハードウェアの規定はEMSの仕様にないため，メモリボードのメーカーは各社で独自の機構(たとえばページを切り替えるためのIOポートアドレスなど)をもっている．したがって，それを制御するソフトウェア(EMSドライバ)も，当然ながらそのメモリボード用のものを使わなければならない．A社のメモリボードが安価だが，EMSドライバはB社のものが使いやすい……ということでも，この２つを組み合わせて使うことは基本的にできないわけだ．

この方式の利点は，第１にプロテクトモードなどのCPU機能を使用しないため，8086やV30搭載機でも使用できるという点だ．また，ページをハード的に切り替えるため，高速な処理が

図3



メモリ管理用語集
アドレッシングモード　メモリアドレスの参照方法．アドレスの指定をレジスタを通さず，オペランドで直接参照する方法を直接アドレッシングモードという．

可能になる．

ソフトウェアEMS

80286以降のCPUは，「リアルモード」と「プロテクトモード」という2つのモードをもっている．「リアルモード」が8086CPUとの互換モードで，アクセスできるメモリアドレスの範囲は1MB以内なのに対し，プロテクトモードは従来の1MBの壁を超えて拡張メモリ領域をアクセスできる．ソフトウェアEMSは，このプロテクトモードを利用して，1MB以降のメモリを各論理ページに割り振って使用し，必要になった論理ページをページフレームにコピーする．

順を追って整理してみると

①プロテクトモードに移行する
②該当の論理ページを1MB以降のメモリからページフレームにコピーする
③リアルモードに復帰する

こういう手順になる．

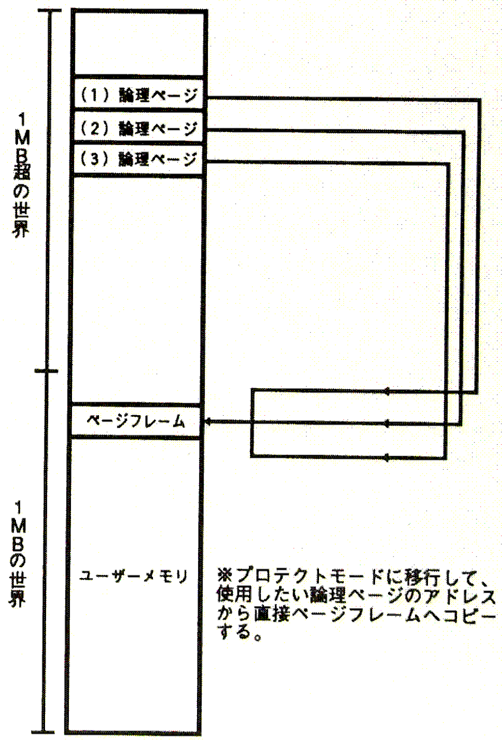
この方式を使えば，メモリボード自体は単純なプロテクトモード用のメモリを使用することができ，何の仕掛もいらない．A社のボードを使ってB社のドライバを組み込むことも当然可能だ．

ただ，メモリボード自体に何の仕掛もいらないということは，ページフレームに使用する領域には最初からRAMが存在していなければならないということになる．たいていのマシンはページフレームに使えるRAMを余分にはもっていないので，ユーザーエリアを削ってページフレームに使うという結果になる．EMSでエリアを拡張するためにユーザーエリアを削るというのは少々矛盾があるようだが．

PC-9801では，ハードウェアに仕掛を作って，グラフィックVRAMの裏側にページフレーム用のRAMを配置して，切り替えて使うという機構をもっている機種がある．これならたしかにユーザーエリアを削ることなくソフトウェアEMSが実現できるが，グラフィックVRAMとの切り替えということで，ドライバは機種依存するし，何よりもEMSとグラフィックVRAMの表示の切り替えに非常なロスが生じるため，この手法は普及できなかったようだ．

また，ハードウェア切り替えなら一瞬で終わるページフレームへの切り替えを，ソフトウェアでコピーするという手順を踏むため，実行速度が遅くなる．そのうえ，286CPUはいったんプロテクトモードに移行すると，リアルモードに戻るためにはCPU自体にリセットをかけなければならないという恐ろしい仕様になっているので，そのための処理時間もたっぷりと必要になる．このため，ひんぱんにページフレームを読み換えるようなソフトウェアでは，ちょっと実用的な速度を出すの

図5
ソフトウェアEMSの原理



メモリ管理用語集
アボート　例外の命令の番地が知られていないトラップ．

が難しく，現状ではあまり利用されていないようだ.

仮想86EMS

仮想86モードを使ったEMSは，386CPU以上を搭載した機種のみで利用することができる．386CPU以降のCPUでは，286のプロテクトモードに加えて，仮想86モードという機能が追加されている．この機能を概念的に図式すると図6のようになる.

ここで図6のいちばん上にある「386システム」というのが，メモリ空間すべてを見通して，それぞれに1MBの世界を構築し管理する核プログラムだと考えてほしい．1MBの世界の中は，あたかもリアルモードで動作しているかのような環境に見えている(実際，この中で動くアプリケーションは，まったく仮想86モードを意識しなくてもよい)．しかも，仮想86モードもプロテクトモードの一種なので，1MBの世界の中で割り込みやI/Oなどの命令が発生したときに，386システムがそれを検出して適切な介入をすることができる．Windows 3.0の386エンハンスドモードは，この機能を使って複数のDOSアプリケーションを動作させているという.

もう1つ，仮想86EMSに必要な386CPUの機能がある．それは「ページング」と呼ばれる機能だ．仮想86モードでは，任意の場所に1MBの世界が構築できるのだが，MS-DOSの1MB空間は，物理アドレス0から始まっていることが前提なので，このままではまずいことになる．そこで386CPUの機能である「ページング」を使用することになる．「ページング」は大まかにいうと，「物理アドレスを4KBごとに区切って，物理アドレスと論理アドレスの対照テーブルを作成し，任意の論理アドレスとして割り付ける」という機能だ．386システムから見ると，物理的には連続していないメモリを，対照テーブルをうまく作ることによって，1MBの世界の中ではあたかも連続した領域であるかのように見せかけることができるのだ．また，このテーブルの内容を書き換えてやれば，瞬時にしてまったく別の4KBの領域と入れ替えることも可能になる.

これで想像がつくかと思うが，仮想86EMSでは物理ページの部分の対照テーブルを書き換えることによって，任意の論理ページを割り付ける操作をしている．対照テーブルを書き換えるだけなので，ソフトウェアEMSのようなコピーの必要はまったくないうえに，物理的にはRAMが存在していないアドレスに物理ページを割り付けることも可能になる.

いいことばかりのような仮想86EMSだが，実際にはプロテクトモード下で動作しているので若干の注意が必要である．たとえば，自分自身でプ

図6
仮想86モードの概念

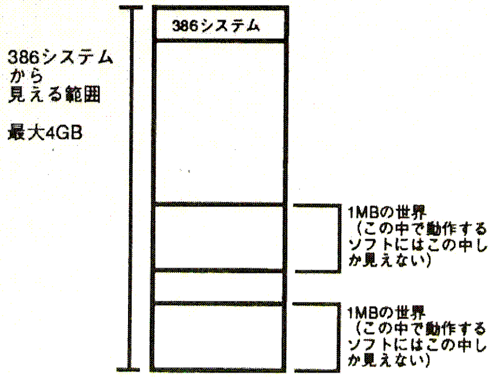
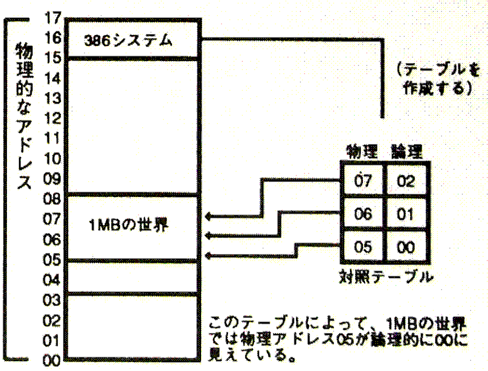


図7
ページングの概念



メモリ管理用語集
アレイ（array）　配列のこと．ファンクションリクエストで情報配列の取得は，そのポインタをレジスタにセットすることにより行われる.

ロテクトモードに移行して，拡張メモリを操作するようなアプリケーションを使うと正しく動作しない場合がある．とはいっても，一般的なプログラムを動かしているぶんには，こうした心配はしなくても386CPUの恩恵を享受することができるだろう．

EMS仕様のバージョンについて

前述のように，EMSの規格は1985年にその原形がLotus Development社とIntel社によって発表された．この発表の直後にMicrosoft社が加わり，Lotus-Intel-Microsoft 3社の頭文字を取って「LIM Ver.3.2」の仕様となる．LIM Ver.3.2の仕様では，

- 物理ページ（ページフレーム）は連続した4ページ（16×4＝64KB）で固定
- 論理ページは8MBまで

となっていた．この仕様を大幅に拡張したのが，現在主流となりつつある「LIM Ver.4.0」の仕様だ．LIM Ver.4.0では，

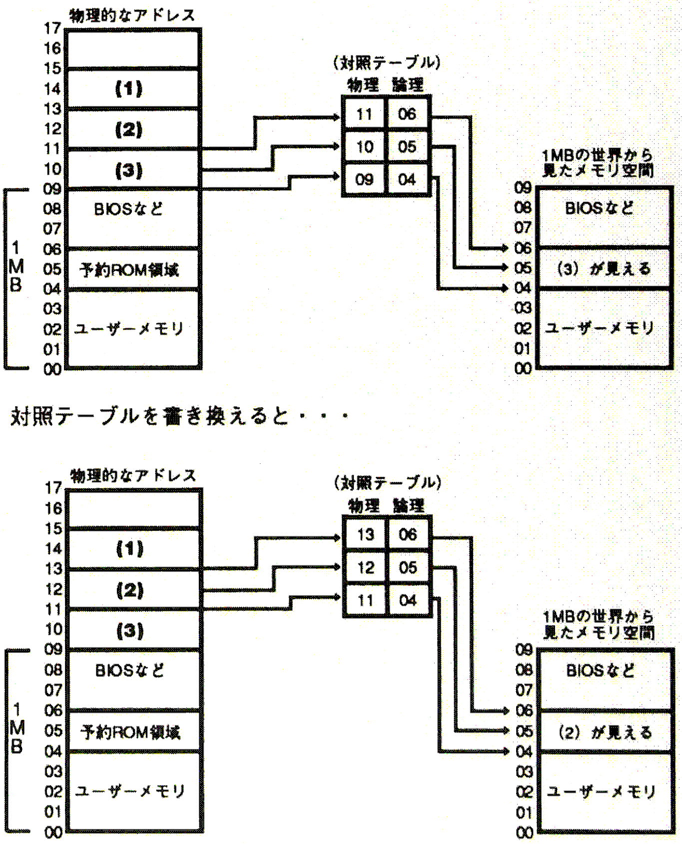
- 物理ページはユーザーエリアの上位アドレスに最大12ページまで任意に設定可能
- 論理ページは32MBまで

とかなり大幅な機能アップが図られている．

このように大幅に機能強化されたLIM Ver.4.0だが，ここでLIM Ver.3.2との互換性の問題が出てきた．LIM Ver.3.2はページフレームを固定で連続4ページ(64KB)確保していたのだが，LIM Ver.4.0では必ずしもページフレームが連続しているとは限らない．また，アプリケーションは各ページの状態を格納するバッファを用意することになっているが，4ページ固定の仕様を前提としてバッファを固定長で用意してしまうと，5ページ以上のページフレームを確保できるLIM Ver.4.0で動作させたときに，バッファがあふれるという問題が発生したりする．LIM Ver.3.2の規格を厳密に守ってプログラム設計をやっていれば，互換性の問題は出てこないはずなのだが，とくに日本ではこうした「決め打ち」のソフトウェアが散見される．

LIM Ver.4.0のもう1つの特徴は，ユーザーエリア（640KBの中）にも最大24ページまでのページフレームを設定することができるようになったことだが，この機能はいまのところMS-DOS自身やWindowsなどのシステム側のプログラムにしか使用を許され

図8
仮想86EMSの原理



メモリ管理用語集
アロケート（allocate）　EMMはアプリケーションからメモリの要求があった場合，論理ページをアロケート(割り当て)する．

ていない．ユーザーエリアの中に24ページのページフレームを確保すると，プログラムの切り替えなどにかなり有効に使えるのではないかと思うが，現在のところこの機能をMS-DOSが使用しているようすはない．将来的な拡張として期待される．

現行のEMS各方式の長短

前述したEMS 3方式の長短を表1にまとめてみた．

表1

	長所	短所
ハードウェアEMS	ハード的に切り替えるので高速 ドライバのロジックがシンプル 8086以上ならどのCPUでもOK	専用のボードが必要 ドライバとボードの組み合わせが限定される
ソフトウェアEMS	メモリボードに特別の仕掛けが不要	ソフトでコピーするため遅い 286以上のCPUが必要 物理的なRAM領域にページフレームが必要
仮想86EMS	メモリボードに特別の仕掛けが不要 コピーを伴わないので高速	386CPU以上でないと使えない 動作原理が複雑

XMS (eXtended Memory Specification)

XMSは，Windows 2.1とともに世に知られるようになった．当初はWindows以外でその名を聞くことはあまりなかったが，ここのところXMSを利用する日本語入力プロセッサが見られるようになってきた．

XMSというと，後述する「1MB直後の約64KBを使えるようにする機能」だけのように思われているふしもあるが，実際のXMS仕様にはこのほかに「UMB」，「EMB」と呼ばれる機能がサポートされている．ただ，現在使用されているXMSドライバでこの3機能をすべて利用できるようになっているものは少ないようだ．

HMA (High Memory Area)

この機能は，80286以上のCPUの不思議な(?)現象を利用している．8086 CPUのメモリ空間は1MBなのだが，これを16進数で記述すると，

00000h～FFFFFh

となる．さて周知のように8086のアドレスは16ビットずつのセグメントとオフセットで指定するので，その指定方法で書くと，

0000:0000～FFFF:000F

となる．つまりオフセットは16ビット（0～FFFFh）なのだが，最後のFFFFhセグメントの場合だけは000Fhで終わりというわけだ．

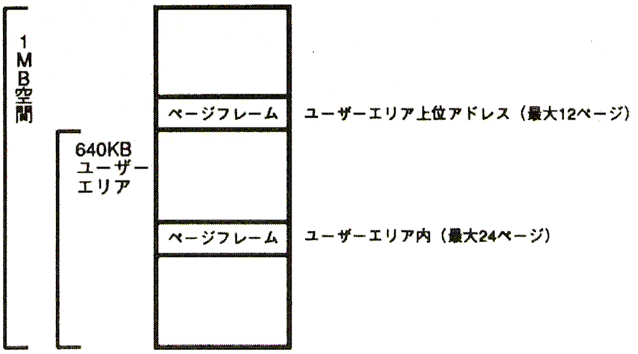
ところが，プロテクトモードをもつ80286CPU以降は，このFFFFhセグメントの場合もオフセットがFFFFhまで指定できるようになって（しまって?）いる．

0000:0000(00000h)～FFFF:FFFF(10FFEFh)

の空間がリアルモードからアクセスできる結果になるのだ．この100000hから10FFEFhまでの約64KBにRAMが実装されていれば，リアルモードからアクセスできることになる．この領域を「HMA」と呼んでいる（図10）．

ところで，8086CPUではアドレスFFFFFhを＋1すると00000hとなる

図9
LIM Ver.4.0でのメモリマップ



メモリ管理用語集
EMS (Expanded Memory Specification)　ロータス・インテル・マイクロソフトの3社が提唱した拡張メモリの仕様．1985年に3.2版，1988年に4.0版が公開された．

（1周して元に戻ってしまうというイメージで考える）．これは，8086CPUのアドレス空間が20ビットしかないので，

FFFFFh+1=100000h

この部分しか認識できないためだ．

通常MS-DOSマシンは，80286以上のCPUを搭載していても，8086との（つまり，MS-DOSマシンとしての）互換を保つうえで，この21ビット目以降のアドレスをアクセスできないようにしている．ただし，これでは1MB超のアドレスを指定することができないので，プロテクトモードに移行する際は，この21ビット目以降のアドレスを有効にするようなハードウェアの仕掛がされている．

HMAを使用するときも，結果的に100000h以降のアドレスを指定することになるので，リアルモードなのにもかかわらず同様な操作が必要になる．

では，XMSドライバはHMAを具体的にどう管理するかというと，

- HMAの使用許可を求める呼び出しがくると，すでに使用中かどうかチェックする．
- もし使用中ならエラーを返す．
- 未使用なら使用開始処理として100000h以降がアクセスできるようにハードウェア操作を行う．

このことからわかるように，XMS管理下でHMAを使用できるのはただ1つのアプリケーションだけだ（まぁ，たった64KBたらずなので複数のアプリケーションが分け合って使うというのもどうかと思うが）．ただ，この管理は単に「お約束」で，使用を許可したアプリケーション以外からのアクセスを物理的に禁止するわけではない．現在の環境下ですべてのアプリケーションがXMSの規約を守って動作しているということが前提となっているわけなのだ．

このように，HMAの領域はFFFF:0010(100000h)～FFFF:FFFF(10FFEFh)までの約64KB(正確にいうと64KBマイナス16バイト)である．「たった」64KBと考えられなくもないが，全部で640KBしかなかったエリアが約10パーセント増加するのだから，それはそれで価値があるともいえる．とくにこのHMAは，使用許可さえもらってしまえば，リアルモードからめんどうな操作やファンクションコールをせずに，自由にアクセスできる領域なので，この点では貴重なエリアといえよう．

UMB (Upper Memory Block)

8086CPUは1MBのアドレス空間をもっているが，たいていのMS-DOSマシンは「メモリ容量640KB」という仕様になっている．では残りの部分はどうしたかというと，いわゆる「BIOS」と呼ばれる部分や画面表示のためのVRAM，そして拡張用に予約されたエリアなどに使用されている．

図10 HMA

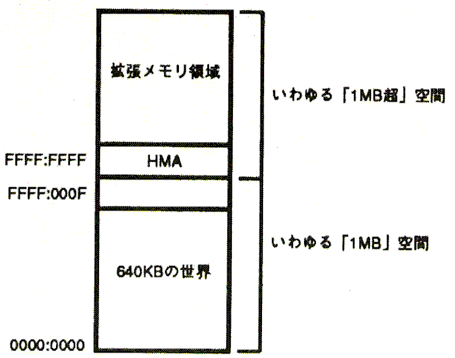
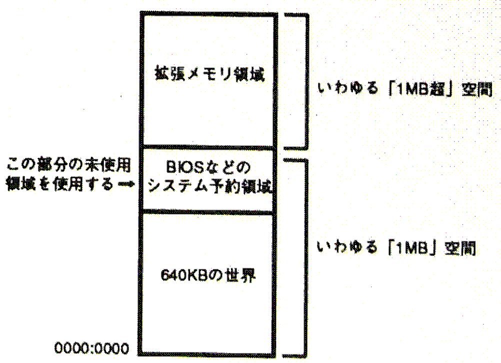


図11 UMB



メモリ管理用語集
EMSエミュレータ　仮想8086モードで80386のページング機能を使ってEMSのページマッピングをエミュレートする．

図12
EMB

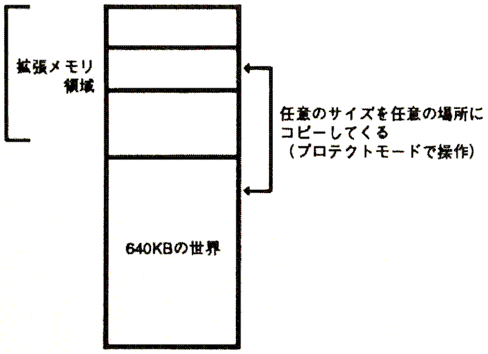
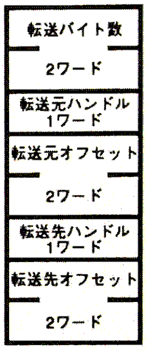


図13
EMBを使用するときのパラメータブロック



UMBの原理は「システムとして予約されているが，現在は使っていない領域」をユーザーメモリとして使ってしまおうというものだ．ただ，この部分は機種によって空き状況が違ってくるのみならず，個々のマシンの拡張状態によっても異なってくるので，標準化した仕様を決めることはできない．ユーザーが自分のマシンの状態をみて，どこからどこまでを使うのか決めるということになる．

また，この部分は拡張用に予約されているだけで，RAMが実装されているわけではないので，UMBとして使いたいアドレスに，新たにRAMを割り付けることのできる機構をもったハードウェアを追加する必要がある．最近は，386CPUの仮想86モードを利用してプロテクトメモリをこの部分にマッピング（仮想86EMSと同じ原理）するドライバも出てきている．こちらのほうは，すでに利用されている向きも多いのではないだろうか．

いずれにしても，リアルモードで直接アクセスすることのできる1MB空間の中は1バイトでも無駄に遊ばせておくのはもったいないという，少々セコい話ではあるが，MS-DOSのメモリ制約との戦いはこれだけシビアであるということだろう．

EMB (Extended Memory Block)

EMBは，どちらかというとソフトウェアEMSに近い．つまり，1MB以降の拡張メモリをプロテクトモードを利用してアクセスし，リアルモードで処理可能な1MB以内の領域とデータの交換をする機能だ．

ソフトウェアEMSと異なるのは，アプリケーションが任意の位置に任意の大きさで「窓」を作成できる点だ．アプリケーションは，まず希望するメモリブロックの大きさをパラメータにして，XMSドライバへメモリ獲得要求を出す．もしその大きさのブロックが取れるだけの空き領域が拡張メモリにあれば，XMSドライバは管理テーブルを更新して，そのブロックにハンドルをつけたうえでアプリケーションに通知する．アプリケーションからは，そのブロックが物理的にどのアドレスに取られたかを知ることはできない（知ることができたとしても，アプリケーションが動作しているリアルモードからはアクセスすることのできない領域なのだが）．実際にそのブロックをアクセスするときには，与えられたハンドルをパラメータとして，1MB以内のエリアとそのブロックの間のデータ交換を行う．

アプリケーションがEMBを実際にアクセスするときは，図13のようなパラメータブロックを作成してXMSドライバを呼び出す．

XMSドライバは，受け取ったパラメータブロックの中のコピー元およびコピー先のアドレスのハンドルをチェックして，それが「0」であれば1MB以内の実アドレスであると判断し，ハ

メモリ管理用語集
EMM　EMSを利用するためのソフトウェア．ハードウェア方式EMSメモリボードはメーカーごとにハードが異なるが，EMMによって同じファンクションを提供する．

ンドルまたは実アドレスをプロテクトモード下でのアドレスに変換する．そして，プロテクトモードに移行して指定バイト数分のコピーを実行し，終了すると再びリアルモードに復帰してアプリケーションに戻っていく．

EMBはEMSと異なり，任意のサイズで任意の場所に拡張メモリとの交換ができるという利点はあるが，ソフトウェアEMSと同様に速度的な面でやや難点があることは否めない．

EMB機能は，従来ほとんど利用しているアプリケーションが存在していなかったが，Windows 3.0付属のディスクキャッシュドライバ「SmartDrv」などは，XMSドライバを利用してメモリを獲得しているようで，この機能を使用しているものと思われる．

メモリ拡張はどこへ行くのか？

Windows 3.0がしだいにその全貌をあらわしつつある．Windowsは，そのグラフィックスユーザーインターフェースや，統一された操作性といった面ばかり強調される傾向にあるが，忘れてはならないのはアプリケーションに与えられるメモリ空間の広大さだ．スタンダードモード以上の場合には，「Windowsアプリケーション」は最初からプロテクトモード下で起動されるため，640KBなどという制約とは無縁の世界になる．とくに，386CPUでの仮想記憶環境下では，4GBという気の遠くなるような空間がサポートされている（もっとも，4GBの仮想メモリ空間を用意できるハードウェアを構築するためにも，気の遠くなるような投資が必要だが……）．まぁ善しにつけ悪しきにつけ，今後のMS-DOS関連の拡張メモリ利用環境は，この巨大プログラムを中心に動いていくことだろう．

もっとも，純然たるMS-DOS環境のことを考えてみると，従来のMS-DOSアプリケーションは，プロテクトモードで動くことを考慮していたりはしないので，一気にプロテクトモードの世界まで行くことは容易なことではないだろう．やはり当分は，EMS，XMSを中心に，従来のアプリケーションとの共存が可能な拡張方法の改良が進められていくことになろうか．はたまた，6年前には現在のメモリ事情が想像もつかなかったように，数年後には現時点で予想だにしなかったような変革がやってきているかもしれないが……．

コラム2
拡張メモリの使用状況がわかるフリーソフト「MS.COM」

バンク方式，EMS，XMSなど駆け足で紹介してきたが，実際にこれらが自分のシステムでどう組み込まれているかを，手軽に表示できる高機能なフリーソフトウェアがある．筆者も愛用させていただいているK-CRAFT氏作の「MS.COM」を紹介する．

●ソフト名
ms.com v.3.11（1991/08/14現在）

●作者
K-CRAFT氏（NIFTY PFD01265）

●動作環境
MS-DOS汎用（2.xx以上，DOS 5.0，DOS/V，Windows 3.Xにも対応）

●入手場所
Nifty-Serve FGALSTライブラリ8番

●おもな機能

- 640KB以内のユーザーメモリの使用状況表示
- デバイスドライバ使用状況表示
- ファイル使用状況表示
- EMS使用状況表示
- UMB使用状況表示
- BMS使用状況表示
- XMS使用状況表示
- Windows 3.X（エンハンストモード）使用状況表示

たとえば，EMSの使用状況を知りたいときには，

A:¥>MS -E

と起動すると，図Aのような表示が得られる．筆者のある日のメモリ使用状態を表示させてみたが，まだ論理ページには空きがあるのがおわかりになるだろう．

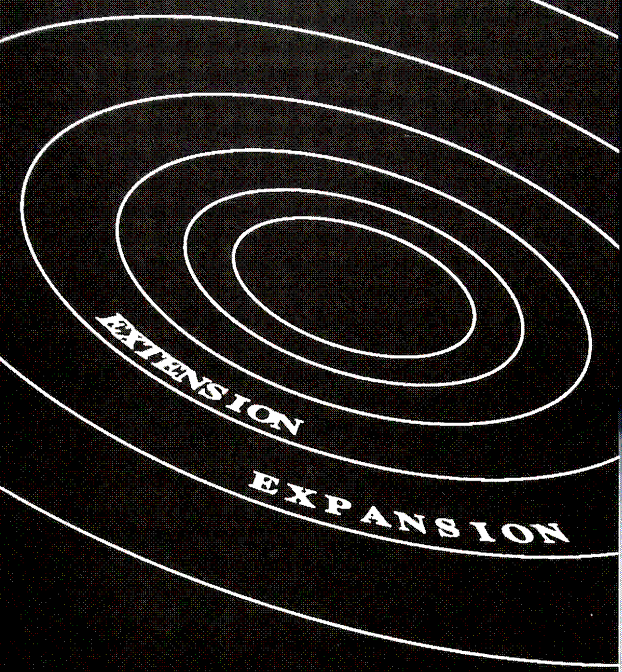
図A

```
EMM version    :  4.0    Total Pages       : 241 ( 3856 KB)
Page Frame     : D000h   Allocated Pages   : 135 ( 2160 KB)
Physical Pages :    4    Unallocated Pages : 106 ( 1696 KB)

Mappable Pages : D000h D400h D800h DC00h

Handle   Name       Pages   Used         Attr
------   --------   -----   ----------   --------
  0                    0    (    0 KB)   [ DRAM ]
  1      WX2K          4    (   64 KB)   [ DRAM ]
  2      WX2           2    (   32 KB)   [ DRAM ]
  3      VZ          128    ( 2048 KB)   [ DRAM ]
  4      FD_HISTS      1    (   16 KB)   [ DRAM ]
```

メモリ管理用語集
EMB（Extended Memory Block）　1MBを超えるメモリ空間のうちHMAを除く領域を指す．



仮想記憶モードをもった高機能CPUの概要

80286, 80386のアーキテクチャ

秋山登

インテル社の8086CPUが発売された当初は，クロック速度も5MHzと，今から比べればあまり速くなく，ソフトウェアもトランスレータで8080のソフトを変換した程度のものしかなく心細いかぎりだった．86系CPUである8088がIBM PCに採用されてからは，パソコンの標準的なCPUとなり，ソフトウェアの数も飛躍的に多くなった．これに伴って，CPU自身も改良が重ねられ，現在では32ビットの486にまで進化してきている．ところが，86系のCPU用に開発されるソフトウェアのほとんどは，MS-DOS上での動作を前提としており，進化したCPUの先進的な機能をあまり利用していないのが現状である．そこで，本節では386CPUを中心に，その先進的な機能とアーキテクチャを紹介する．

86系CPUの概要

80286は16ビットのCPUですが，86系のCPUの中ではじめて，プロテクトモードと呼ばれるセグメンテーションによる仮想記憶管理機構を実現したCPUです．プロテクトモードについてはあとで詳細を説明しますが，8086で採用されたアドレッシングの方法と互換性を最大限に保ちながら，8086CPUでは不足がちであったメモリ空間を拡大するという巧みな方法です．

プロテクトモードには，メモリの保護と仮想記憶を実現するために不可欠な管理機能，タスクスイッチなどの高度な機能も含まれています．しかし，80286CPUには1つのセグメントの大きさが最大64KBという，8086CPUから引き継いだ弱味がありました．この弱味を解決したのが32ビットCPUである386です．386では，80286のプロテクトモードを完全に引き継ぐと同時に，プロテクトモードが32ビットCPUにふさわしいように機能強化され，1つのセグメントの大きさは実メモリ空間と同じ4GBに拡大されました．

386では，さらにページングと呼ばれる仮想記憶管理機構が追加されると同時に，プロテクトモードの1種である仮想86モードも加わりました．これは，現在までに蓄積された膨大な量のリアルモードのソフトウェア資産をさらに有効に活用する道を開くものです．386は，リアルモードと呼ばれる8086と互換性のあるモードで，8086と同一の機械語を実行できますが，それに加えてほとんどのワードオペランドをダブルワードのオペランドとして扱う機能をもっています．リアルモードでは，32ビットのポインタはあまり有効ではありませんが，プロテクトモードでは大量のデータを処理する場合などに有効です．

最後に登場したのが486です．このCPUは，機能的には386+387+8KBのメモリキャッシュといったところですが，処理速度に大きな改善が見られます．以上の特徴を表1にまとめて示します．

レジスタとフラグの構造

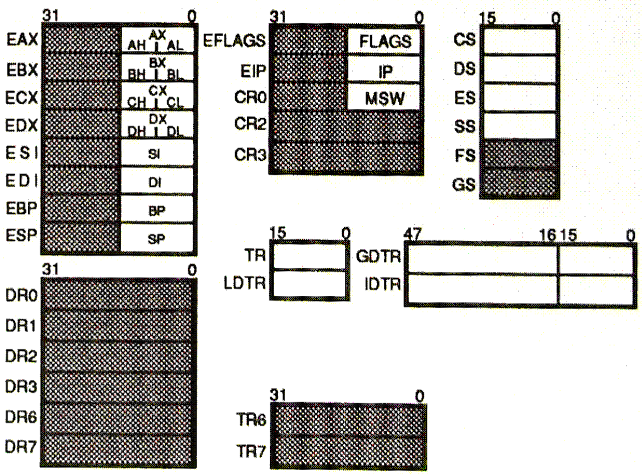
386のレジスタの構成を図1に示します．286CPU以降大きく変わった点は，汎用レジスタが32ビット化された

表1：Intel社の16/32ビットCPUの比較

特徴	8086	286	386	486
リアルモード	○	○	○	○
プロテクトモード	×	○	○	○
仮想86モード	×	×	○	○
ページング	×	×	○	○
32ビット汎用レジスタ	×	×	○	○
32ビットオフセット	×	×	○	○
オンチップメモリキャッシュ	×	×	×	○
オンチッププロセッサ	×	×	×	○
実メモリ空間[MB]	1	16	4096	4096
I/O空間[KB]	64	64	64	64
クロック速度[MHz]（最大値）	10	12.5	33	33

図1

80386CPUのレジスタ構成（網掛けの部分は80286CPUにはない）



メモリ管理用語集
Windows　Microsoftが1985年にVer.1.0を発売した．1990年に発売された3.0はメモリ管理方法が大きな改善をうけ，プロテクトメモリや仮想記憶などもサポートしている．

図2

286/386/486CPUのフラグレジスタ

ビット	31	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
286					0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF
386	予約		VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF
486	予約	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

ことでしょう．EAXを例にとりますと，リアルモードからもプロテクトモードからもEAXとしても，AXとしても，あるいはAH, ALとしてもアクセス可能です．

各種セレクタを保持するレジスタは，80286と同じく16ビットのままです．図1のように386からは，FSとGSという新たなセグメントレジスタが導入されています．内部のプログラムからは見えない，ディスクリプタキャッシュのベースアドレスは32ビットになっています．GDTRとIDTRは，ともに32ビットのベースアドレスに対応しています．80286のFLAGSレジスタは32ビットに拡張され，図2と表2のような新たなビットが誕生しました．RFはDR（デバッグレジスタ）によるブレークフォールトをサポートするため，VMは仮想86モードをサポートするために設けられています．

486のACは，メモリアクセス効率に重大な影響を及ぼす不適当な境界整列を例外として検出するためのものです．

80286のMSWは，32ビットに拡張されてCR0と呼ばれるようになっています．CR0で注目すべき点は，80286でリセットできなかったPEビットがMOV命令を使用すればリセットできるようになったことです（互換性を保つためにLMSW命令ではリセットできない）．これによって，PEビットをセットすることでプロテクトモードに入り，PEビットをリセットすることでリアルモードに復帰することが可能になりました．

486で増加したビットは，オンチップのコプロセッサとメモリキャッシュなどに関するビットです．予約ビットを使用していないソフトウェアに対しては互換性が保たれています．

このほか，本文では詳細は述べませんが386と486CPUにはDR（デバッグレジスタ）とTR（テストレジスタ）があります．DRには，ICEに装備されているような，メモリアクセスに関するブレークポイントを4つまで設定することができます．結果は，デバッグフォールトとして例外により報告されます．TRは，ページング機構のTLB（トランジションルックアサイドバッファ）をテストするため使用することができます．486ではTR3～TR5が追

表2：286/386/486CPUのフラグ一覧

ビット	フラグ	機能	286	386	486
0	CF	キャリー	○	○	○
2	PF	パリティ	○	○	○
4	AF	補助キャリー	○	○	○
6	ZF	ゼロ	○	○	○
7	SF	符号	○	○	○
8	TF	シングルステップ割り込み	○	○	○
9	IF	割り込み許可	○	○	○
10	DF	自動増加/減少の方向	○	○	○
11	OF	オーバーフロー	○	○	○
12/13	IOPL	I/O 特権レベル	○	○	○
14	NT	タスクのネスト	○	○	○
16	RF	ブレークポイントの無視	×	○	○
17	VM	仮想86モード	×	○	○
18	AC	境界整列チェック	×	×	○

図3

MSW/CR0のビット構成

ビット	31	30	29	18	17	16	5	4	3	2	1	0
MSW (286)							予約		TS	EM	MP	PE
CR0 (386)	PG	予約						R	TS	EM	MP	PE
CR0 (486)	PG	CD	NW	予約 AM	予約	WP	予約 NE	1	TS	EM	MP	PE

加されていて，これらのレジスタを使用することにより，メモリキャッシュのテストを行うこともできます．

新たな命令とアドレッシングモード

80286，386，486で追加された命令は，汎用の命令を強化するもの，プロテクトモードに関するもの，新しい機構に対処するものです．表3に示すように386では，アドレスサイズのプリフィックスとオペランドサイズのプリフィックスが導入されています．これらのプリフィックスにはニーモニックがなく，アセンブラがソースコードの分脈から判断して自動的に付加します．

リアルモードでは，これらのプリフィックスが現れると，プリフィックスのつけられた命令にかぎり，アドレスサイズあるいはオペランドサイズが32ビットに変更されます．両方のプリフィックスを同時に利用することもできます．

プロテクトモードでは，実行中のコードセグメントのデフォルトサイズによって，16ビットになったり32ビットになったりします．アドレスサイズとオペランドサイズがともに16ビットの場合は，8086/286と互換性のある命令となります．アドレスサイズを32ビットにすると，あとで述べるような，32ビットのオフセットを使用する新しいアドレッシングモードを使用することになります．また，オペランドサイズを32ビットにすると，ワードオペランドはほとんどの場合，ダブルワードのオペランドとして処理され，32ビットの汎用レジスタへのアクセスも可能になります．

表3：8086にない新たな命令（○：あり，×：なし）

命　令	オペランド	286	386	486
ARPL	r,r/m,r	○	○	○
Address Size Prefix		×	○	○
BOUND	r,m	○	○	○
BSF/BSR	r,r/r,m	×	○	○
BSWAP	r	×	×	○
BT/BTC/BTR/BTS	r,r/m,r/r,i/m,i	×	○	○
CLTS		○	○	○
CMPXCHG	r,r/m,r	×	×	○
ENTER	i,i	○	○	○
FS:/GS:		×	○	○
IMUL	r,r/r,m/r,i	×	○	○
IMUL	r,r,i/r,m,i	○	○	○
INS/OUTS		○	○	○
INVD		×	×	○
INVLPG	m	×	×	○
Jcc	d32	×	○	○
LAR	r,r/r,m	○	○	○
LEAVE		○	○	○
LFS/LGS/LSS	r,m	×	○	○
LGDT/SGDT	m	○	○	○
LIDT/SIDT	m	○	○	○
LLDT/SLDT	r/m	○	○	○
LMSW/SMSW	r/m	○	○	○
LSL	r,r/r,m	○	○	○
LTR/STR	r/m	○	○	○
MOV	CRn,r/r,CRn	×	○	○
MOV	DRn,r/r,DRn	×	○	○
MOV	FS GS,r m/r m,FS GS	×	○	○
MOV	TRn,r/r,TRn	×	○	○
MOVSX/MOVZX	r,r/r,m	×	○	○
Operand Size Prefix		×	○	○
POP/PUSH	FS/GS	×	○	○
POPA/PUSHA		○	○	○
PUSH	i	○	○	○
RCL/RCR	r,i/m,i	○	○	○
ROL/ROR	r,i/m,i	○	○	○
SAL/SAR/SHR	r,i/m,i	○	○	○
SETcc	r/m	×	○	○
SHLD/SHRD	r,r,i CL/m,r,i | CL	×	○	○
VERR/VERW	r/m	○	○	○
WBINVD		×	×	○
XADD	r,r/m,r	×	×	○

表3を眺めていて，とくに目立つことは，386ではビット関係の命令が強化されたことでしょう．全体的な命令強化の方向は，モトローラ社の68000系のCPUを意識していると思えます．

486では，この表のほかに当然のことながら387の命令が使用できます．

命令について見落としてはならないのは実行速度です．表4に，よく使用されると思われる命令の実行に必要なクロック数をあげてあります．最初の8086はクロックが5MHzでしたから，1クロック当たり200nsの時間がかかりました．33MHzの486では約30nsです．

たとえば，表のCALL命令で比較してみると，

メモリ管理用語集
XMS（eXtended Memory Specification）　80286以上のCPUで1MBを超える空間にマッピングしたメモリの管理方式．HMAとEMBに分けられる．

表4：よく使用される命令の実行に必要なクロック数

命　令	オペランド	8086	286	386	486
ADD	R,R	2	2	2	1
CALL	d16	19	7+m	7+m	3
Jcc	d8	16/4	7+m/3	7+m/3	3/1
MOV	AX,mem	10	5	4	1+2
MOV	mem,AX	10	3	2	1
RET	(NEAR)	8	11+m	10+m	5

注　m：次に実行する命令のバイト数
／：ジャンプする場合／ジャンプしない場合
＋：キャッシュのミスヒットが発生した場合

表5：32ビットアドレスモードの実効アドレス指定

アドレッシングモード				デフォルトセグメント(除レジスタ)
変位なし	8ビット変位付	32ビット変位付	レジスタ	
[EAX]	d8[EAX]	d32[EAX]	AL/AX/EAX	DS:
[ECX]	d8[ECX]	d32[ECX]	CL/CX/ECX	DS:
[EDX]	d8[EDX]	d32[EDX]	DL/DX/EDX	DS:
[EBX]	d8[EBX]	d32[EBX]	BL/BX/EBX	DS:
			AH/SP/ESP	
DS:d32	d8[EBP]	d32[EBP]	CH/BP/EBP	SS:
[ESI]	d8[ESI]	d32[ESI]	DH/SI/ESI	DS:
[EDI]	d8[EDI]	d32[EDI]	BH/DI/EDI	DS:
[EAX+ix*s]	d8[EAX+ix*s]	d32[EAX+ix*s]		DS:
[ECX+ix*s]	d8[ECX+ix*s]	d32[ECX+ix*s]		DS:
[EDX+ix*s]	d8[EDX+ix*s]	d32[EDX+ix*s]		DS:
[EBX+ix*s]	d8[EBX+ix*s]	d32[EBX+ix*s]		DS:
[ESP+ix*s]	d8[ESP+ix*s]	d32[ESP+ix*s]		SS:
DS:[d32+ix*s]	d8[EBP+ix*s]	d32[EBP+ix*s]		SS:
[ESI+ix*s]	d8[ESI+ix*s]	d32[ESI+ix*s]		DS:
[EDI+ix*s]	d8[EDI+ix*s]	d32[EDI+ix*s]		DS:

注　ix=EAX,ECX,EDX,EBX,EBP,ESI,EDI,なし　s=1,2,4,8

```
200×19=3800[ns]  (8086)
 30× 3=  90[ns]  (486)
```

となり，486のほうが8086より40倍以上速いことになります．表にあげたクロック数は，最適条件のときの値です．最適条件はCPUによって異なるので，このような単純な比較は実際の場合必ずしも成立しませんが，プログラマにはあまり目立たないCPUの心臓部の改良も着実に進んでいるといえます．

新たな命令に加えて，386からは32ビットのアドレスサイズで表5のような対称性のよい強力なアドレッシングモードが使用できます．

たとえば，

```
MOV    EAX, [ESI]
ADD    EAX, [EDX+EDI*4]
```

という具合です．表5の[]の中に実名で現れるレジスタはベースレジスタと呼ばれます．表の中でixと記述されているレジスタはインデックスレジスタと呼ばれ，上の例ではADD命令の第2オペランドのEDIがこれに該当します．インデックスレジスタには，ESPを除く汎用レジスタが指定できます．*sはスケールファクタと呼ばれ，実効アドレスを計算するときに，インデックスレジスタの値をs倍して計算します．sには1，2，4，8を指定することが可能です．

これらのアドレッシングモードでは，32ビットのオフセットが使用されますから，4GBの実メモリ空間すべてをカバーでき，プロテクトモードでは事実上メモリアクセスに対する制限が取り除かれたことになります．ただし，リアルモードや仮想86モードでは32ビットのアドレッシングモードは使用できますが，16ビット以上のオフセットを指定することはできません．

プロテクトモード

プロテクトモードは前に述べたように，80286から導入されたセグメンテーションと呼ばれる仮想記憶管理の手法を提供するモードです．セグメンテーションでは，メモリはセグメントと呼ばれる論理的なデータの集合として扱われます．換言すれば，メモリはセグメント単位で管理されます．この管理には，特権レベルのチェック，用途のチェック，存在のチェック，セグメントの大きさのチェックなどが含まれています．そして，チェックにパスしないときは，例外が発生してセグメントを不法に侵害されるのを防いだり，仮想記憶ソフトウェアを起動したりします．

たとえば，あるプログラムのコード部分，初期化されたデータ部分，初期化されないデータ部分，参照はされるが書き換えられることのないデータ部分などは，それぞれ1つの独立したセグメントを構成します．セグメントのデータをアクセスするには，セグメントを選択するセレクタとセグメント内

の位置を特定するオフセットから構成されるポインタを使用します．

それでは，このポインタとリニアアドレスの関係からプロテクトモードを説明することにします．リニアアドレスという耳慣れない言葉がでてきましたが，プロテクトモードの説明では実アドレス(物理アドレス)と同じものと考えてください．

ポインタとリニアアドレス

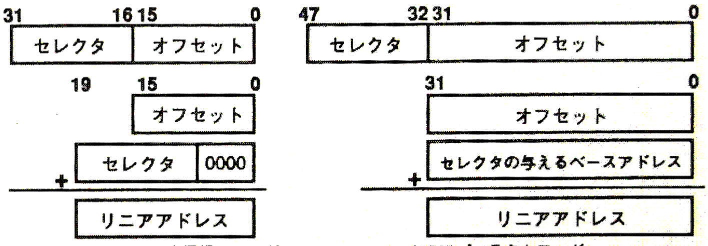
リアルモードでは，よくご存知のようにセレクタすなわちセグメントレジスタの値は，リニアアドレスと直接関係があります．セレクタを16倍したものが，セグメントの先頭のリニアアドレスすなわちベースアドレスになり，これにオフセットそのままを加算すると，ポインタをリニアアドレスに変換することができます．この関係は仮想86モードでも同じです．

ところがプロテクトモードでは，セレクタとリニアアドレスは直接関係がありません．セレクタは，アドレッシングモードのメモリ間接のような働きをします．プロテクトモードのセレクタは，GDTRおよびLDTRにおかれたベースアドレスから始まるGDTおよびLDTと呼ばれるテーブルからベースアドレスを求めます．そして，そのベースアドレスにオフセットを直接加算することによって，ポインタをリニアアドレスに変換します．このようすを図4に示します．

このアドレス変換のようすをもう少し詳細に説明しましょう．

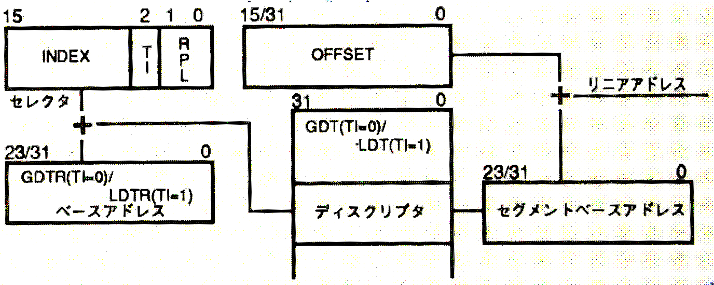
セレクタは，図5に示すような構造をしています．RPLは，要求特権レベルと呼ばれ0から3の値をとります．

図4
各モードにおけるポインタとリニアアドレスの関係



(a)リアルモード/仮想86モード　(b)386プロテクトモード

図5
ポインタからリニアアドレスへの変換



これは，特権レベルのチェックにCPUが使用します．TIはセレクタからベースアドレスを求めるときに，GDTを参照するかLDTを参照するかを決めるビットです．GDTの場合0，LDTの場合1です．CPUのレジスタGDTRとLDTRには，LGDTあるいはLLDT命令で初期化された，それぞれのテーブルのベースアドレスとテールの大きさを示すリミットが入っています．GDTとLDTには，あとで述べるディスクリプタと呼ばれる特別な形式の8バイトの大きさのデータが入っています．セレクタ部のINDEXは，それぞれのテーブルの何番目かのディスクリプタを指定することになります．指定されたディスクリプタの中に記述されているベースアドレスが，ポインタのオフセットに加算されてリニアアドレスになります．

以上の過程を図5に示しました．80286の場合，オフセットが16ビット，ベースアドレスが24ビットであること以外，386/486とまったく同様です．なお，各テーブルのリミットを越えるようなセレクタが指定された場合には例外が発生します．

コードとデータセグメントのディスクリプタ

通常のコードとデータのセグメントに関するディスクリプタの形式は図6のとおりです．80286の場合，最初の6バイトだけが使用されます．残りのバイトは，すべて0であるとみなされます．386/486では，8バイトすべての情報を使用します．これは，以下に説明するディスクリプタについても同様で

メモリ管理用語集
エンハンスドモード　Windows 3.0において80386以降のCPUで1MBを超えるメモリ空間，仮想メモリ，複数のMS-DOSアプリケーションを起動可能にするモード．

す．ディスクリプタの中のBASEは，先ほど説明したセグメントのベースアドレスです．

6バイト目のGビットは，グラニュルと呼ばれLIMITの単位を指定します．G=1であればLIMITの単位は4KB，G=0であればLIMITの単位は1バイトです．Dビットは，セグメントのデフォルトのサイズを示します．D=1で32ビット，D=0で16ビットです．Dビットは，コードセグメントとスタックセグメントに対して重要な意味をもちます．コードセグメントでは，デフォルトのアドレスサイズとオペランドサイズを決定します．スタックセグメントでは，PUSH/POPされるデータの暗黙のサイズを決定します．Xビットはユーザーが自由に使用できるビットです．

第5バイト目のPビットは，セグメントが実際に存在するかどうかをCPUに知らせるビットです．P=1で存在することを示し，P=0で不在であることを示します．CPUがP=0のセグメントを参照しようとすると，セグメント不在の例外を起こし，この例外を利用して，OSは仮想記憶機構を作動させることができます．DPLは2ビットで0～3の値をとり，ディスクリプタの特権レベルを示します．Aビットは，ディスクリプタがアクセスされると1にセットされるビットで，仮想記憶の割り当てなどに利用できます．

LIMITは20ビットでG=0なら1MB，G=1なら4GBまでのセグメントの大きさ，すなわちオフセットの最大値を指定できます．なお，LIMITで指定される大きさは，実際のセグメントサイズ-1ですから，LIMITに0を指定するとG=1のとき4KB，G=0のとき1バイトの大きさのセグメントを指定したことになります．

図6

コード／データセグメントのディスクリプタ

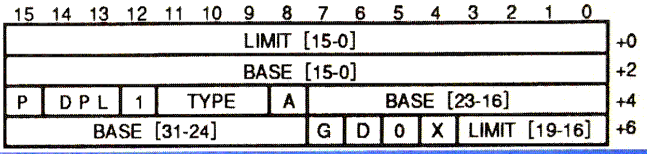


表6：コード／データセグメントのタイプと性質

TYPE	セグメントの性質	ロード可能なセグメントレジスタ
0	リードのみ	DS, ES, FS, GS
1	リードとライト	DS, ES, FS, GS, SS
2	リードのみ，下方伸長	DS, ES, FS, GS
3	リードとライト，下方伸長	DS, ES, FS, GS, SS
4	実行のみ	CS
5	実行とリード	CS, DS, ES, FS, GS
6	実行のみ，コンフォーミング	CS
7	実行とリード，コンフォーミング	CS, DS, ES, FS, GS

表7：システムセグメントとゲートのタイプ

TYPE	ディスクリプタの種別	関連する命令	内容
0	未定義		
1	286TSS，利用可能	CALL,JMP,LTR	TSSセグメントディスクリプタ
2	LDT	LLDT	LDTセグメントディスクリプタ
3	286TSS，使用中	(IRET/NT=1)	TSSセグメントディスクリプタ
4	286コールゲート	CALL,JMP	コードセグメントへのポインタ
5	タスクゲート	CALL,JMP,割り込み,例外	TSSセグメントセレクタ
6	286割り込みゲート	割り込み，例外	コードセグメントへのポインタ
7	286トラップゲート	割り込み，例外	コードセグメントへのポインタ
8	未定義		
9	386TSS，利用可能	CALL,JMP,LTR	TSSセグメントディスクリプタ
A	未定義		
B	386TSS，使用中	(IRET/NT=1)	TSSセグメントディスクリプタ
C	386コールゲート	CALL,JMP	コードセグメントへのポインタ
D	未定義		
E	386割り込みゲート	割り込み，例外	コードセグメントへのポインタ
F	386トラップゲート	割り込み，例外	コードセグメントへのポインタ

最後に残ったTYPEフィールドは，表6のようなセグメントの性質を表します．表中の下方伸長とは，スタックセグメントのように，オフセットの低い方向へ利用されるセグメントです．ほかのセグメントと異なり，LIMITの値はオフセットの下限値であると解釈されます．

コンフォーミングとは，どの特権レベルからも呼び出せる例外的なコードセグメントです．コンフォーミングセグメントにあるコードは，呼び出した側の特権レベルで実行されます．システムで共通に利用するルーチンを置くセグメントとして適しているといえます．

システムセグメントのディスクリプタ

システムセグメントのディスクリプ

図7
システムセグメントのディスクリプタ

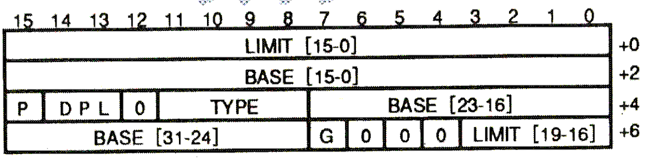


図8
ゲートディスクリプタ

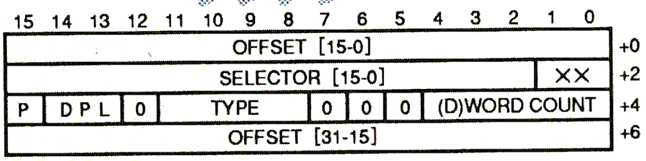


図9
コールゲートによるセグメント間のコールとジャンプ

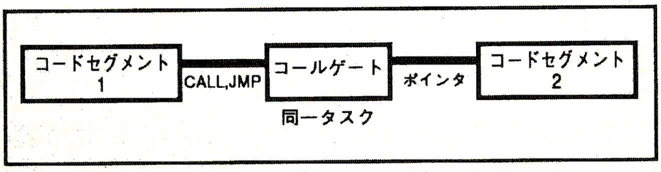
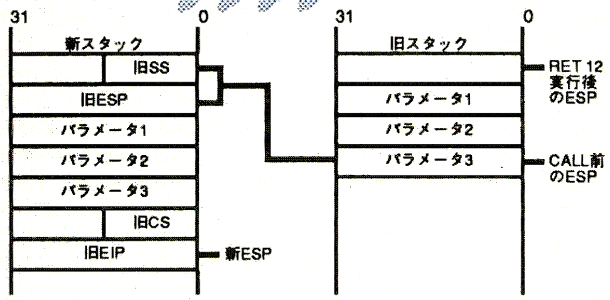


図10
スタックの切り替えとパラメータの複写（コールゲート）



タは，前記のコード／データセグメントのディスクリプタと同じ形式です．ただし，6バイト目のDビットとXビットはありません．システムセグメントは，LDTとTSSの2種類でTYPEによって指定します．TYPEフィールドの値は表7にあります．LDTセグメントはシステムに任意の個数置くことができます．タスクごとに置くことも，システムに1つだけ置くことも，あるいはまったく置かないこともできます．TSSセグメントは，タスクスイッチに使用されるセグメントで，マルチタスクシステムではTCBと同様に利用されます．TSSディスクリプタは，GDTにだけ配置することが可能です．

ゲートディスクリプタ

ゲートディスクリプタは，図8のような構造のポインタでメモリ間接的なジャンプやコールを実現します．リアルモードの割り込みベクタテーブルの1要素のようなものです．TYPEフィールドの値によって，表7のような種類が存在します．

Pビット，DPLフィールドは，それぞれ存在とディスクリプタの特権レベルを示し，コード／データセグメントのディスクリプタと同様です．(D) WORD COUNTフィールドは，コールゲートでのみ有効です．特権レベルの遷移が発生したときには，あとで述べるようにスタックが切り替えられますが，このとき以前のスタックから新しいスタックに複写すべきパラメータの(D) WORD数を指定します．386コールゲートのとき，このフィールドに指定された数値がダブルワード単位で，286コールゲートのときワード単位で複写されます．ゲートの場合，それに含まれるセレクタのRPLフィールドはつねに無視されます．

コールゲート

コールゲートは，システムサブルーチンのエントリポイントをユーザーに指示する目的などで使用されます．コールゲートを使用できる命令は，FAR CALLとFAR JMPに限られます．これらのCALLあるいはJMP命令は，オペランドにオフセットとセレクタをもっていますが，セレクタがコー

メモリ管理用語集
オーバーレイ　640KBのメモリ空間で，より大きなプログラムを実行するための方法．実行モジュールを必要に応じてディスクから呼び出す．

ルゲートを指定していれば，それらの命令のオフセットは使用されないで，ゲートの中にあるオフセットが使用されます．図９に示すように，コールゲートの中に置かれているポインタがCALLあるいはJMP先となります．

コールゲートでは，FAR CALLを行った場合に限って，特権レベルの遷移を行うことができます．この場合，図10のように，スタックの切り替えが行われます．スタックの切り替えが行われた場合には，つねに新しいスタックに以前のスタックのSSとESPの値がプッシュされます．このとき，ゲートの(D)WORD COUNTフィールドが０でなければ，以前のスタックから指定された数のパラメータがコピーされます．

図10では，386のコールゲートでパラメータの数を３と指定した場合のようすを示してあります．286コールゲートでは，プッシュされるすべての要素が16ビットのものとなります．図10の状態でコールゲートによって呼び出されたルーチンが終了して，RETF 12という命令を実行すると，双方のスタックから３つのダブルワードのパラメータが削除されます．

図11

TSSの構成

a)286TSS

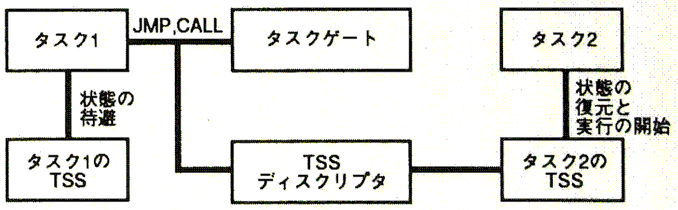
15　　　0	
バックリンクセレクタ	+0
特権レベル0用SP	+2
特権レベル0用SS	+4
特権レベル1用SP	+6
特権レベル1用SS	+8
特権レベル2用SP	+10
特権レベル2用SS	+12
IP	+14
FLAGS	+16
AX	+18
CX	+20
DX	+22
BX	+24
SP	+26
BP	+28
SI	+30
DI	+32
ESセレクタ	+34
CSセレクタ	+36
SSセレクタ	+38
SSセレクタ	+40
LTDセレクタ	+42

b)386TSS

31	0	
0	バックリンクセレクタ	+0
特権レベル0用ESP		+4
0	特権レベル0用SS	+8
特権レベル1用ESP		+12
0	特権レベル1用SS	+16
特権レベル2用ESP		+20
0	特権レベル2用SS	+24
CR3		+28
EIP		+32
EFLAGS		+36
EAX		+40
ECX		+44
EDX		+48
EBX		+52
ESP		+56
EBP		+60
ESI		+64
EDI		+68
0	ESセレクタ	+72
0	CSセレクタ	+76
0	SSセレクタ	+80
0	DSセレクタ	+84
0	FSセレクタ	+88
0	GSセレクタ	+92
0	LDTセレクタ	+96
I/Oビットマップオフセット	0 　T	+100
1ビットをI/O空間アドレス1バイトに対応させたIO許可ビットマップ（最大8KB+1バイト（OFFH））1=禁止、0=許可		+IOビットマップオフセット

図12

JMP/CALL命令によるタスクスイッチ



タスクスイッチ

タスクスイッチは，TSSセグメントあるいはタスクゲートをオペランドのセレクタとしたFAR CALLあるいはFAR JMPで起こります．これらの命令のオペランドのオフセットは無視されます．あるいは，FLAGSのNT=1の状態でIRET命令を実行することによっても起こります．

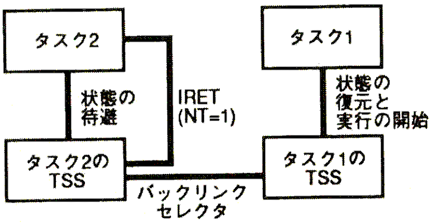
タスクスイッチに使用されるTSSセグメントは，286か386で図11のように異なります．どちらのタスクゲートにも，特権レベル０～２までのスタックの値が記録されています．コールゲートで述べたように，特権レベルの遷移があると，これらの値にスタックが初期化されます．386TSSのI/Oビットマップオフセットは，TSSによって指定されるタスクがIOPLより低い特権レベルのとき使用できる，I/O空間を指定するビットマップへのオフセットを指定します．このビットマップは，１ビットがI/O空間の１バイトに対応しており，対応するビットが０のときアクセス許可，１のときアクセス禁止を示します．

たとえば，ビットマップの最初のバイトのビット０がI/O空間の0000H

メモリ管理用語集

拡張メモリ　バンクメモリやEMSメモリのように直接アドレス空間にマッピングせずにメインメモリ空間にページフレームを開き，ページ単位で割り当てアクセスする．

図13
IRET命令によるタスクスイッチ



に対応します．同じく，386TSSのオフセット100にあるTビットはデバッグのためのもので，1にセットしておくと，タスクスイッチ完了後最初の命令を実行する前に，デバッグトラップの例外を発生させることができます．

FAR CALLとFAR JMPによってタスクスイッチが起きると，以前のタスクは実行が凍結されます．このために，以前のタスクのTSSにタスクの状態が保存されます．このとき保存される状態は，386TSSではEIP～GS，286TSSではIP～SSです．CR3とLDTは保存されません．そして，新たなタスクのTSSから新たなタスクの状態がCPUにロードされます．ロードされる状態は，386TSSではCR3～LDT，286TSSではIP～LDTです．このときFAR CALLでタスクスイッチが起きているならば，元のタスクに復帰できるように，新しいTSSのバックリンクフィールドに以前のTSSのセレクタ値を書き込み，FLAGSのNTを1にします．FAR JMPによるタスクスイッチでは，バックリンクフィールドには書き込みを行わないでNTを0にします．そして以前のタスクのTSSディスクリプタのTYPEフィールドを利用可能にセットします．

いずれの命令で起きたタスクスイッチでも，新たなタスクのTSSディスクリプタのTYPEフィールドは使用中にセットされます．また，TRには新たなタスクのTSSセレクタがロードされます．これによって，実行中のタスクのTSSはつねにアクセス可能になります．

このようなTSSのタスクスイッチのメカニズムはリエントラントでないので，使用中のタスクにスイッチすることはできません．以上のタスクスイッチのようすを図12に示します．

FLAGSのNTビットが1の状態で，IRET命令を実行してもタスクスイッチが起こります．このときは，以前に実行していたタスクに戻ることになり，図12とは動作が異なります．NT=1の状態でIRETを実行すると，タスクの状態を凍結するため，CALLやJMP命令のときと同じように，タスクの状態を実行中のタスクのTSSにセーブします．そして，バックリンクフィールドのセレクタの値を使用して，戻るべきタスクのTSSをアクセスして，CALLやJMP命令のときと同じ内容の状態を復元して，中断されていたタスクの実行を開始します．当然のことですが，IRETで実行を凍結されたタスクのTSSのTYPEフィールドは利用可能にセットされます．

特権レベルに関する規則

CPUがディスクリプタを使用するときは，つねに特権レベルのチェックが行われます．このとき，特権レベルの比較がDPLとの間で行われますが，その1つの基準となるのがCPLと呼ばれる特権レベルです．CPLは，現在実行中のコードセグメントのもつ特権レベルで，CSセレクタのRPLフィールドの値と一致します．

特権レベルには，4つのレベル（0～3）があります．特権レベル0が一番特権が高く(強く)，特権レベル3が一番特権レベルが低い(弱い)と定義されています．以下の規則では，混乱を避けるために特権レベルの比較はレベルを表す数値で行うことにします．

①データに対するアクセス
　DPL>= CPL AND DPL>= RPL
②コードに対するアクセス（ゲートを使用する場合を除く）
　JMPおよびCALL命令の場合
　　DPL = CPL AND DPL>= RPL
　コンフォーミングセグメントへのCALL命令の場合
　　DPL<= CPL
③コールゲートを使用する場合
　JMP命令の場合
　　ゲートのDPL>= CPL AND
　　ゲートのDPL>= RPL AND
　　目的コードのDPL = CPL
　CALL命令の場合
　　ゲートのDPL>= CPL AND
　　ゲートのDPL>= RPL AND
　　目的コードのDPL<= CPL
　コンフォーミングセグメントへのCALL命令の場合

メモリ管理用語集
仮想86マネージャ　プロテクトモードでハードウェアページング機能を利用し，仮想メモリを実現するプログラム．

ゲートのDPL>= CPL AND
ゲートのDPL>= RPL AND
目的コードのDPL<= CPL

④スタックセグメントへのディスクリプタのロード
DPL = CPL AND DPL = RPL

⑤コンフォーミングセグメントからのデータリード
DPL，CPL，RPLの値による制限なし

⑥タスクスイッチの場合
TSSセグメントへのCALL，JMP命令の場合
TSSのDPL>= CPL AND
TSSのDPL>= RPL
タスクゲートを使用したCALL，JMP命令の場合
ゲートのDPL>= CPL AND
ゲートのDPL>= RPL

特権命令とFLAGS変更の制限

特権命令は，実行できる条件によって表8のように2種類に分類することができます．リアルモードでは暗黙の特権レベルは0ですが，この表の内のCLTS，LTR，MOV命令を実行する

表8：特権命令とI/O保護命令

命　　令	実行できる条件
CLTS	CPL=0
HLT	CPL=0
LGDT	CPL=0
LIDT	CPL=0
LMSW	CPL=0
LTR	CPL=0
MOV CRn,r/r,CRn	CPL=0
MOV DRn,r/r,DRn	CPL=0
MOV TRn,r/r,TRn	CPL=0
CLI	IOPL>=CPL
STI	IOPL>=CPL
IN	IOPL>=CPL OR BIT MAP=0
INS	IOPL>=CPL OR BIT MAP=0
OUT	IOPL>=CPL OR BIT MAP=0
OUTS	IOPL>=CPL OR BIT MAP=0

表9：特殊な扱いを受けるフラグ

フラグ	変化させることのできる条件
VM	CPL=0 and IRET
IOPL	CPL=0 and POPF,IRET
IF	CPL<=IOPL and CLI,STI,POPF,IRET

表11：割り込みの優先順位

優先順位	割り込み，例外のタイプ
1	デバッグフォールト
2	デバッグ以外のフォールト
3	INT 命令
4	デバッグトラップ
5	NMI 割り込み
6	INTR 割り込み

表10：例外の種類と区分

割り込み番号	例外の原因	例外の区分	エラーコード	80286	80386	80486	リアルモード
0	除算エラー	フォールト	なし	○	○	○	○
1	デバッグ	フォールト／トラップ	なし	○	○	○	○
2	NMI 割り込み	NMI	なし	○	○	○	○
3	INT3割り込み	トラップ	なし	○	○	○	○
4	INTO 割り込み	トラップ	なし	○	○	○	○
5	BOUND 命令	フォールト	なし	○	○	○	○
6	不当命令コード	フォールト	なし	○	○	○	○
7	NDP 利用不可	フォールト	なし	○	○	○	○
8	2重フォールト	アボート	あり	○	○	○	○
9	NDP セグメントオーバーラン	アボート	なし	○	○	×	○
10	不当な TSS	フォールト	あり	○	○	○	×
11	セグメント不在	フォールト	あり	○	○	○	×
12	スタックフォールト	フォールト	あり	○	○	○	×
13	一般保護	フォールト	あり	○	○	○	○
14	ページフォールト	フォールト	あり	×	○	○	×
16	NDP エラー	フォールト	なし	○	○	○	○
17	不適当な境界整列	フォールト	あり	×	×	○	×

仮想86モード　80386以上のCPUが持つ動作モードで，仮想マシンモニタが実際のメモリ管理や入出力・割り込みを処理し，アプリケーションは8086上と同様に動作する．

ことはできません．また，SMSW，SGDT，SIDT命令以外のプロテクトモードに関する命令を実行することもできません．

リアルモードで，これらの実行できない命令を実行しようとすると，不当命令コードの例外が発生します．また，プロテクトモードで実行できる条件以外の条件で実行すると一般保護例外が発生します．

FLAGSレジスタの特定のビットは，表9の条件に適合する条件下でないと変化させることができません．条件に適合しない状態で，POPF命令やIRET命令で，これらのビットを変化させようとした場合には，例外は発生しませんが，これらのビットは変化しません．

割り込みと例外

プロテクトモードとリアルモードで発生する例外を表10にまとめて示しました.例外の区分のうち,フォールトは原因となった命令を実行する以前の状態が保存されていて，原因を取り除いたあとで再実行できる可能性のあるものです．トラップとは，原因となる命令を実行したあと発生する例外です．トラップが発生したときには，原因を究明することはできませんが，次の命令から実行が再開できます．アボートは，再実行するための情報が保存されていない例外で，通常，処理を中止する以外に方法はありません．486で新設された不適当な境界整列の例外は，EFLAGSのACビットがセットされているとき，特権レベル3からのみ発生します．

割り込みや例外が同時に発生したときは，表11の優先順位にしたがって割り込み，例外が発生します．たとえば，デバッグフォールトとデバッグトラップが同時に発生した場合には，デバッグフォールトが最初に発生し，その処理の終了後デバッグトラップが発生します．

図14は，プロテクトモードでの割り込みや例外発生時の処理の流れを示したものです．リアルモードのように，ベクタの位置は固定されていません．LIDT命令で指定されたベースアドレスから始まるIDTと呼ばれるテーブルに，格納されたディスクリプタが目的の割り込み処理の位置を示しています．割り込みや例外のベクタ番号は，IDTに格納されたディスクリプタの先頭からの位置を決定します．IDTに配置することのできるディスクリプタは，タスクゲート，割り込みゲート，

図14
割り込み，例外発生時の制御の流れ

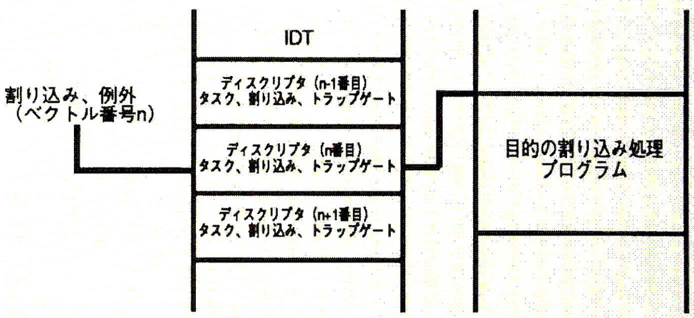


図15
割り込み時にスタックに待避される内容（特権レベルの遷移とエラーコードのある場合）

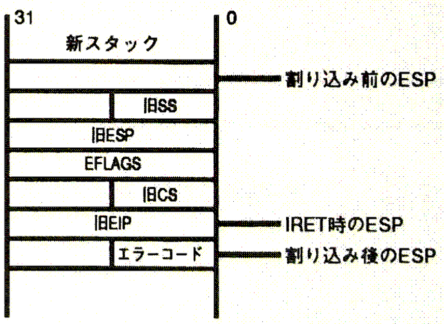


表12：仮想86モードで追加される特権命令

命令	例外が発生する条件
INT n	IOPL<3
IRET	IOPL<3
POPF	IOPL<3
PUSHF	IOPL<3

メモリ管理用語集
仮想86モニタ　仮想86モードを使ってEMSメモリなどを実現するためのプログラム．一般にはデバイスドライバの形式で提供される．

トラップゲートに限られます．ゲートのDPLは，INT nとINTOによるソフトウェア割り込みの場合のみチェックされ，そのほかの割り込みや例外の場合には無視されます．

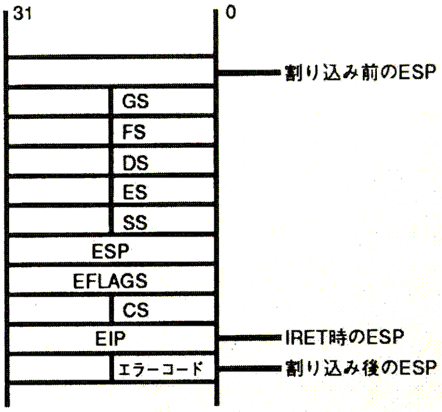
割り込みゲートとトラップゲートでは，コールゲートと同様にして制御が流れます．ただし，割り込みから正常に回復するために，FLAGSのNTビットはIRET命令に備えて0にされます．これは，NTビットが1であるとタスクスイッチが行われるためです．また，割り込みゲートでは割り込みが禁止されますが，トラップゲートでは割り込み許可の状態は割り込まれる以前の状態のまま保存されます．タスクゲートの場合には，CALL命令で行われるものと同様のタスクスイッチが行われます．

割り込みゲートとトラップゲートによる割り込み処理への移行の過程で，特権レベルの遷移が発生した場合には，図15に示すような情報が新たに使用されるスタックにプッシュされます．図15に示されているとおり，エラーコードのある場合には，割り込み処理を終了してIRET命令を実行するときにスタックから取り除いておくことが必要です．これは，特権レベルの遷移を伴わない場合も同様です．

仮想86モード

仮想86モードは，特権レベル3のプロテクトモードの1種です．プロテクトモードから仮想86モードへの移行は，TSSのEFLAGSフィールドのVMビットを1にしてタスクスイッチを行うことで達成できます．ポインタの扱いがリアルモードとまったく同じなので，割り込み処理に注意すれば，通常のリアルモードのソフトウェアをそのまま実行することが可能です．ここでは，リアルモードと通常のプロテクトモードにない，例外的な事項を説明します．

図16
仮想86モードにおける割り込み時のスタック



仮想86モードとプロテクトモードの差異の第1点は，IOPLの扱いが異なることです．IOPLは，表12のFLAGSを操作する命令に対してのみ作用します．これらの命令をIOPL =3以外で実行すると，エラーコード0の一般保護例外が発生します．

第2点目の相違は，割り込みと例外の制御の流れです．仮想86モードでの割り込みや例外は，モードをプロテクトモードに切り替えます．このとき，スタックには図16に示す内容がプッシュされます．プッシュされるセグメントレジスタは，すべて仮想86モードの値をもっています．そして，これらの値は割り込み処理を行うプロテクトモードでは使用できないので，割り込み処理が開始される前にDS~GSのセグメントレジスタにヌルセレクタがロードされます．ヌルセレクタは，GDTの最初のディスクリプタを指定する値が0のセレクタですが，特殊なセレクタでセグメントレジスタにロードするときは例外を起こしませんが，これを使用してデータをアクセスすると例外が発生します．そこで，割り込み処理でこれらのセレクタを使用する前に，さらに有効なセレクタをロードしなければなりません．また，割り込み処理から仮想86モードに戻って処理を継続するためには，割り込み処理プログラムを特権レベル0で実行させ，IRET命令では32ビットのEFLAG全体を復元します．このように割り込み処理プログラムを実行させると，IRET命令はDS~GSのセグメントレジスタもポップします．

第3点目は，仮想86モードではリアルモードと同様にEIPは1MBまでの空間しか指示できない点です．このため，1MB以上のリニアアドレス上で仮想86モードを動作させるためには，次に述べるページングの機構を使用する必要があります．

メモリ管理用語集
仮想記憶　実際にはマッピングされていないメモリ空間にあたかもメモリがあるようにする．実際にはメモリとディスクをスワップして実現する．

図17
ページングの概念

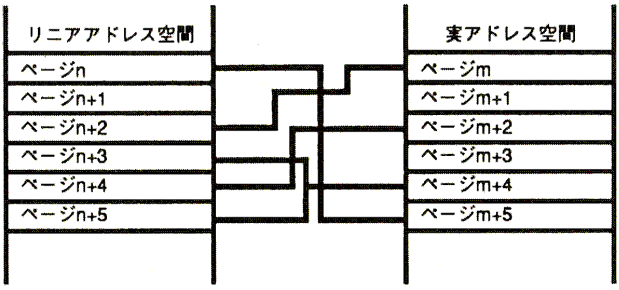


図18
ページングのメカニズム

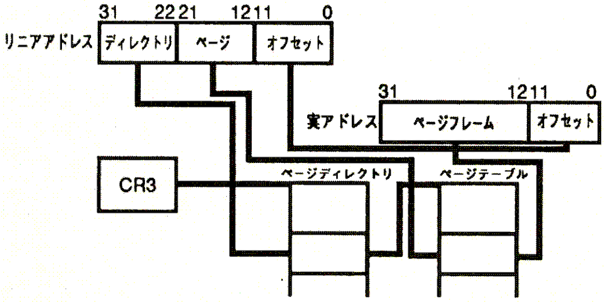


図19
CR2とCR3の形式

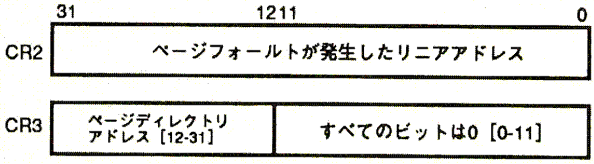
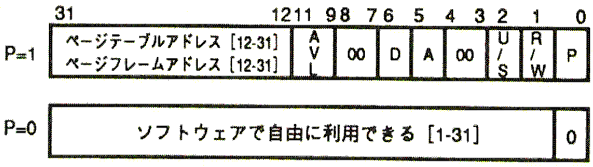


図20
ページディレクトリとページテーブルの項目



【ページング】

ページングは，汎用大型計算機で普通に使用されている仮想記憶の管理手法の1つです．386では，命令コードがセグメンテーション向きになっていますが，セグメンテーションの弱点はセグメントのサイズが一定しないために，メモリに大きなフラグメンテーションが発生しやすく，仮想記憶の管理が困難であることです．この欠点をうまく補う手法が，ここで述べるページングです．

ページングは，メモリ空間をページと呼ばれる一定のサイズに区切って管理します．そして，CPUから見た連続なアドレス空間，すなわちリニアアドレス空間をページごとに実アドレス空間にマッピングします．このような機構を実現すると，ソフトウェアからマッピングを変更するだけで，リニアアドレス空間のページを実アドレス空間のどこにでも配置することができます．そして，メモリが不足する事態になったなら，ある基準で決定した実アドレス空間にあるメモリのページの内容を補助記憶に待避して，その場所にリニア空間で不足しているページをマッピングすることで，仮想記憶を実現することができます．メモリのフラグメンテーションは，ページサイズより少なくなりますので，ページングのページサイズは2KBと4KBがよく使用されています．386ではページサイズは4KBです．

386のページングの機構は，CR0のビット31をセットすると動作を開始します．ただし，リアルモードでは動作することができません．プロテクトモードか仮想86モードにかぎります．ページングの機構では，リニアアドレスは，10ビットのディレクトリと10ビットのページおよび12ビットのオフ

メモリ管理用語集
間接アドレッシング　メモリアドレスの指定をオペランドとして指定するのではなくいったんレジスタなどに格納してから指定する方法

セットに分割されます．

図18にあるように，CR3がページディレクトリの先頭アドレス(実メモリアドレス)をポイントするよう設定します．ページング機構は，リニアアドレスを実メモリアドレスに変換するために，リニアアドレスのディレクトリ部分をインデックスに使用して，ページディレクトリからページテーブルの先頭アドレス(実メモリアドレス)を求めます．そして，次にリニアアドレスのページをインデックスに使用して，ページテーブルから実メモリアドレスのページの先頭であるアドレスページフレームを求めます．これに，リニアアドレスのオフセットを加算すると，リニアアドレスに対応する実メモリアドレスが求められます．ページディレクトリとページテーブルはメモリ上にありますから，この内容を変更することにより，容易にマッピングを変更することができます．

図19は，ページディレクトリとページテーブルの項目の形式です．Pビットは，その項目がポイントするページテーブル，あるいはページフレームが実メモリ上に存在することを示します．P＝0のときは，ページング機構にとってそのほかのビットは無意味で，ソフトウェアによって実メモリの待避先を示す補助記憶のアドレスなどに自由に使用することができます．R/WとU/Sは，ページレベルの保護に使用されます．R/Wは，CPL＝3でリードだけのアクセスが許されるか，リードとライトの両方が許されるかを指定するビットです．U/Sは，CPL＝3でページが使用できるかできないかを示します．この２つのビットによる保護の対象は表13にまとめてあります．Aはアクセスされると１にセットされるビットで，Dは書き込まれると１にセットされるビットです．AとDのビットは，仮想記憶を割り当てるアルゴリズムにとって有用です．３ビットのAVLフィールドは，ソフトウェアで自由に使用できるフィールドです．ページテーブルアドレスおよびページフレームアドレスは，それぞれページテーブルまたは実メモリ上のページの先頭を指示します．

386のページングに関係するレジスタは，CR0のほか，図18に現れたCR3とCR2です．ページレベルの保護に違反したりページの不在が検出されると，ページフォールトと呼ばれる例外が発生します．CR2は，この例外が発生する元になったリニアアドレスを保持しています．CR3は，ページディレクトリの先頭の実メモリアドレスを指定するために使用します．

以上，386CPUを中心にそのアーキテクチャの簡単な概要について説明しました．何分にも膨大な説明を要する機能を一口で述べましたので，詳細については以下の参考文献を参照してください．筆者としては，ここに述べたような高度な機能を使いこなせるプログラマが１人でも多くなることを願ってやみません．

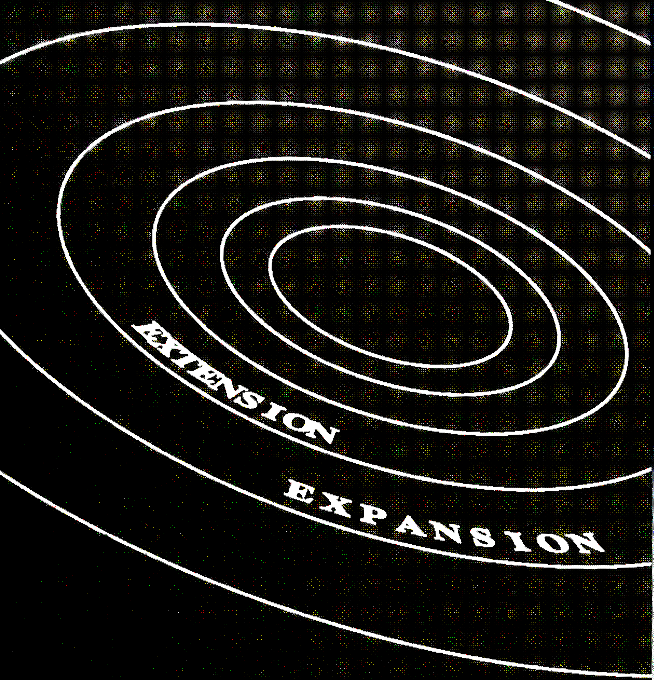
表13：ページのアクセス権

U/S	R/W	CPL<3	CPL＝3
0	0	リード，ライト，実行	アクセス不可
0	1	リード，ライト，実行	アクセス不可
1	0	リード，ライト，実行	リード，実行
1	1	リード，ライト，実行	リード，ライト，実行

参考文献

１．大貫広幸，田中恵介，蓑原隆共著「80286ハンドブック」アスキー出版局，1985
２．J.Crawford，P.Gelsinger著，岩谷宏訳「80386プログラミング」工学社，1988
３．「Microprocessors」Intel，1990

メモリ管理用語集
キャッシュ　メモリを使って，HDDなどから読み込んだデータとそのポインタを記憶して，同じポインタにアクセスしたときはメモリから読み込むことで実行速度を稼ぐ．



特集

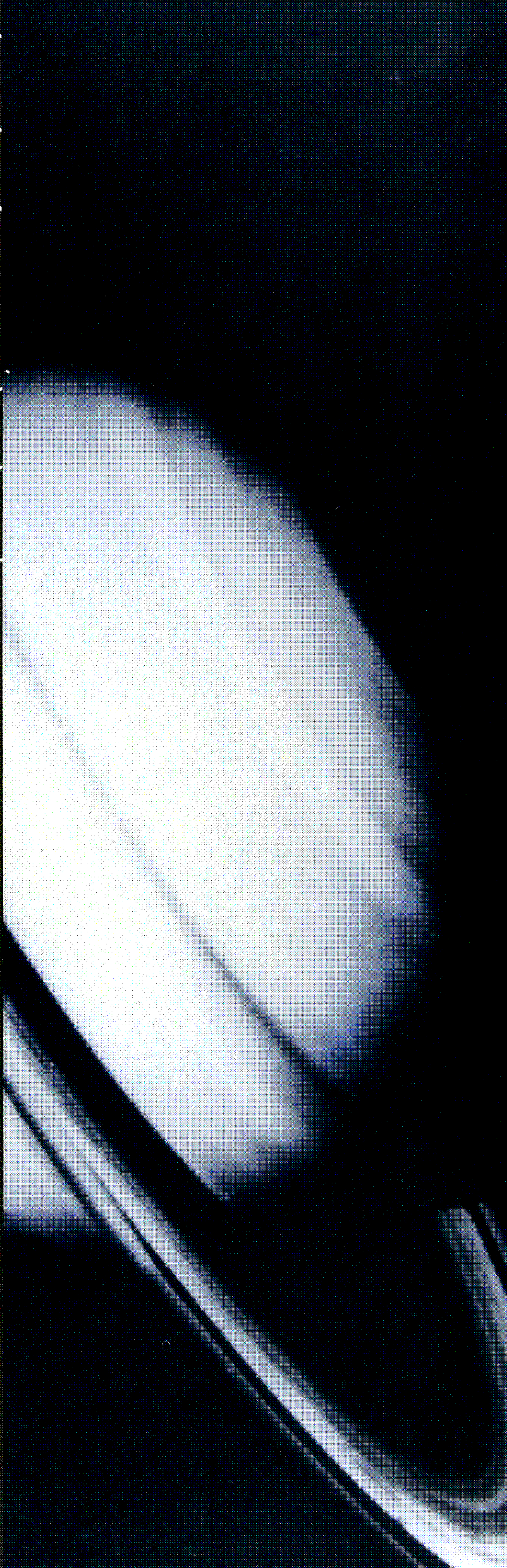
286/386メモリ活用プログラミング

386によるプロテクトモードプログラミングの実際

プログラム実行環境「P386」の作成

秋山登

1982年，インテル社がプロテクトモードをもった80286を発表して以来，そろそろ10年が経過するが，プロテクトモードのプログラミングは未だにあまり盛んになっているとはいい難い．その一因として考えられるのが，手軽な実行環境がないことであろう．世の中のことはさておき，実は筆者自身も似たような状況なので，この特集を機会に簡単な実行環境を作成する決心をした．ここに紹介する実行環境は，プロテクトモードの全機能を取り込んだような大げさなものではない．MS-DOSから立ち上がるプロテクトモードプログラムの単純なローダと，プロテクトモードからリアルモードのソフトウェアにアクセスするためのインターフェースを含んだ単純なプログラムである．名付けてP386．ターゲットハードウェアはPC-9800シリーズの386（DX，SX）搭載機種である．



P386作成の方針

386のプロテクトモードは，リアルモードからみればかなり多くの新しい機能を実現できそうなのだが，あまり欲張ると最後はお手上げといったことになりかねない．そこで必要最低限の機能だけをもたせて，あとはリアルモードのプログラムをプロテクトモードから呼び出して利用するという形式を採用する．その必要最低限の機能というのは，次のようなものである．

①MS-DOSのCOMファイルのような，メモリイメージ形式のプロテクトモードのプログラムをファイルから増設メモリに読み込むローダ．
②ローダによってロードされたプロテクトモードのプログラムを，独立した386タスクとして起動するための初期化機能．
③簡単にするためプロテクトモードの特権レベルは0のみとし，RPLをつねに0として特権のチェックから除く．
④プロテクトモードから，MS-DOSのようなリアルモードのシステムソフトウェアを利用するためのインターフェース．
⑤ハードウェアからの割り込み発生時に，割り込みをリアルモードの割り込み処理プログラムにバイパスさせるようなインターフェース．
⑥例外発生時とプログラム終了時にはプロテクトモードを脱出して，リアルモードから例外の発生した時点，あるいは終了時点のCPUの状態を表示できるようにする．

このような方針にしたがってできあがったプログラムがリスト1である．使用したアセンブラはMASM 5.10，リンカはMS-LINK 5.10である．リスト上に見慣れない表記が少しあるので，それについてここで述べておく．

USE16/USE32

デフォルトのセグメントのサイズ属性をアセンブラに知らせる．これによって，アセンブラはアドレスサイズプリフィックスとオペランドサイズプリフィックスのつけ方を変える．プロテクトモードのコードセグメントでは，このサイズ指定とディスクリプタのデフォルトサイズとが一致していなければならない．また，リアルモードのコードセグメントでは，必ずUSE16を指定しなければならない．さらに，USE32の指定されているデータセグメントを参照する際には，オフセット値がUSE16のセグメントからの参照であっても，自動的に32ビット長にとられることも注意を要する．

Jcc SHORT LABEL

USE32のセグメントでは，Jccのオペランドは32ビットのディスプレイスメントが可能である．LABELが前方参照であると短いディスプレイスメントでも6バイト必要なので，SHORTを指定することにより2バイトの命令を生成させている．

IRETD，LODSD

オペランドの記述でアドレスサイズやオペランドサイズの区別のつく命令は，アドレスサイズやオペランドサイズをオペランドの記述によって指定することができるが，オペランドのない命令では，アドレスサイズやオペランドサイズをこのような新しいニーモニックで指定する．

FWORD PTR

FWORDは，386の6バイト長のポインタにおもに使用されるデータの型である．このプログラムでは，LGDTやLIDT命令などに使用されている．データの定義をDF疑似命令で行えばFWORD PTRの記述は不要である．

OFFSET LABEL

これは新しい機能ではなく，筆者の不精な記述を読んでいただくための説明である．SEGMENT命令にPARAを指定し，かつ結合属性がPRIVATE（何も指定しない）であれば，OFFSET値に再配置が必要であるというマークをアセンブラがつけても，リンカがその値を変えることはない．このような記述がセレクタに関して使われているので注意してほしい．正確には，LABEL-〈セグメントの先頭アドレスを示すラベル〉というふうに書くべきであろう．

セグメントとディスクリプタ

通常のアセンブラのプログラムに比べて，使用するセグメントの数が多いので，表1にその使用目的をまとめてある．この表の中で，セグメント名欄中に()でくくられているものは，ソースコード上のセグメントではなく，MS-DOSを呼び出してメモリを割り当てるものである．

ディスクリプタに関しても表2に用途，セレクタの値などを示した．

表1：セグメントの用途

セグメント名	サイズ	使用目的
TEXT	16	リアルモードのコード，タスク0のコード
INTP	32	プロテクトモード側の割り込み処理
TASK1	32	タスク1のコード
DATA	16	リアルモードのデータ
COM	16	リアルモード／プロテクトモード（レジスタなど）
GDT	16	GDT
IDT	16	IDT
TS0TSS	16	タスク0のTSS
TS1TSS	16	タスク1のTSS
TS2TSS	16	タスク2のTSS
STACK	16	リアルモードのスタック
(LDT)		LDT
(BUF)		リアルモード／プロテクトモード（データ）
(TASK用STACK)		各タスク用のスタック（12領域）

表2：ディスクリプタの一覧

ディスクリプタ	セレクタ	DPL	タイプ	用途
NULL	0000			ヌルセレクタ
GDTDESC	0008	0	DATA R/W	GDTのエイリアス
IDTDESC	0010	0	DATA R/W	IDTのエイリアス
INTDESC	0018	0	CODE RD	割り込み処理
LDTDESC	0020	0	LDT	LDT
COMDESC	0028	0	DATA R/W	パラメータの授受
BUFDESC	0030	0	DATA R/W	データの授受
TS0CODE	0038	0	CODE RD	タスク0のコード
TS1CODE	0040	0	CODE RD	タスク1のコード
TS2CODE	0048	0	CODE RD	タスク2のコード
TS0STK0	0050	0	DATA R/W	タスク0の特権0用スタック
TS1STK0	0058	0	DATA R/W	タスク1の特権0用スタック
TS2STK0	0060	0	DATA R/W	タスク2の特権0用スタック
TS0STK1	0068	1	DATA R/W	タスク0の特権1用スタック
TS1STK1	0070	1	DATA R/W	タスク1の特権1用スタック
TS2STK1	0078	1	DATA R/W	タスク2の特権1用スタック
TS0STK2	0080	2	DATA R/W	タスク0の特権2用スタック
TS1STK2	0088	2	DATA R/W	タスク1の特権2用スタック
TS2STK2	0090	2	DATA R/W	タスク2の特権2用スタック
ST0DESC	0098	0	DATA R/W	タスク0の初期スタック
ST1DESC	00A0	0	DATA R/W	タスク1の初期スタック
ST2DESC	00A8	0	DATA R/W	タスク2の初期スタック
TS0DESC	00B0	0	TSS	タスク0のTSS
TS1DESC	00B8	0	TSS	タスク1のTSS
TS2DESC	00C0	0	TSS	タスク2のTSS
SRCDESC	00C8	0	DATA R/W	作業用
DSTDESC	00D0	0	DATA R/W	作業用

機能構成と処理の概要

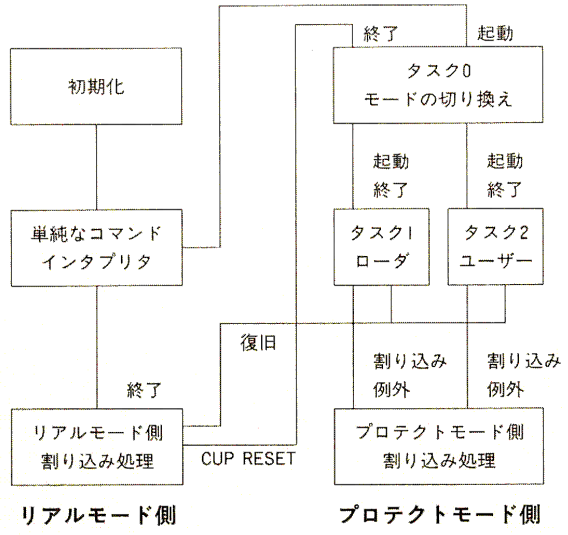
図1は，P386のブロック分けとその関連を示したものである．各ブロックの動作を以下に簡単に述べる．リスト1と併読していただければ幸いである．

初期化

MS-DOSのプログラムとしてP386自身に必要な初期化，プロテクトモードの動作に必要なメモリの確保，GDT，IDT内のディスクリプタの初期化を行う．P386では，将来に備えてLDT，特権レベルの遷移があったときに使用されるスタックを確保している

メモリ管理用語集
シャドウRAM　実行速度の遅いCPUではROMアクセス時にウェイトがかかる．そこでROMと同じアドレスをもつシャドウRAMにROMの内容をコピーし，実行速度を稼ぐ．

図1
プログラムの構成



が，実際には使用していない．大部分の初期化は，リスト1のはじめのほうで行われているが，タスクに関する初期化は，タスクの起動ごとに行う必要があるので，SETUP0～2にサブルーチン化している．

単純なコマンドインタプリタ

コマンドラインの入力と解析を行い，コマンドファイルを読み込む．コマンドにパラメータがあれば，プロテクトモードとリアルモードの間のパラメータの受け渡しのためのセグメント（以下COMセグメント）に設定する．読み込まれたファイルの内容は，タスク0を通じて，タスク1によってタスク2の領域へロードされる．プログラム

リスト1：P386.ASM

```
        PAGE    ,132
        .386P
;
LDLO    EQU     1000H                   ; PROTECTED MODEのPROGRAMを
LDHI    EQU     0001H                   ; LOADするADDRESS/16
STKSIZ  EQU     1000H                   ; 各TASKのSTACK SIZE
;
;       PROTECTED MODEでFAR JUMPの命令を生成させるために使用
;
JMPF    MACRO   OFS,SEL
        DB      0EAH
        DW      OFS,SEL
        ENDM
;
;       PROTECTED MODEでFAR CALLの命令を生成させるために使用
;
CALLF   MACRO   OFS,SEL
        DB      09AH
        DW      OFS,SEL
        ENDM
;
;       CPU RESET後の処理を継続させるための準備
;
RETRST  MACRO
        PUSH    CS
        PUSH    OFFSET RESET
        XOR     AX,AX
        MOV     DS,AX
        MOV     DS:[0406H],SS
        MOV     DS:[0404H],SP
        MOV     BYTE PTR DS:[0481H],80H ; 機種によっては必要
        MOV     AL,0EH                  ; SHUT1=0
        OUT     37H,AL
        JMP     SHORT $+2
        JMP     SHORT $+2
        XOR     AX,AX                   ; 24/32BIT ADDRESS BUS ENABLE
        OUT     0F2H,AL
        JMP     SHORT $+2               ; 機種によっては必要
        JMP     SHORT $+2               ; "
        OUT     0F6H,AL                 ; "
        ENDM
;
;       CPUをS/WからRESETするための命令
;
CPURST  MACRO
        XOR     AL,AL
        OUT     0F0H,AL
        JMP     SHORT $
        ENDM
;
;       PROTECTED MODEでの割込処理のENTRY POINT
;
IENTRY  MACRO
@VCT    =       0
        REPT    255
        PUSH    EAX
        MOV     AL,@VCT
        JMP     INTHND
@VCT    =       @VCT+1
        ENDM
        ENDM
;
TEXT    SEGMENT PARA PUBLIC USE16 'CODE'
TEXT    ENDS
INTP    SEGMENT PARA USE32 'CODE'
INTP    ENDS
TASK1   SEGMENT PARA USE32 'CODE'
TASK1   ENDS
DATA    SEGMENT PARA PUBLIC USE16 'BSS'
DATA    ENDS
COM     SEGMENT PARA USE16 'BSS'
COM     ENDS
GDT     SEGMENT PARA USE16 'BSS'
GDT     ENDS
IDT     SEGMENT PARA USE16 'BSS'
IDT     ENDS
TS0TSS  SEGMENT PARA USE16 'BSS'
TS0TSS  ENDS
TS1TSS  SEGMENT PARA USE16 'BSS'
TS1TSS  ENDS
TS2TSS  SEGMENT PARA USE16 'BSS'
TS2TSS  ENDS
STACK   SEGMENT PARA STACK USE16 'STACK'
STACK   ENDS
;
CR      EQU     0DH
LF      EQU     0AH
MSDOS   EQU     21H
;
;       PROTECTED MODEとREAL MODEとの間の通信用領域
;
COM     SEGMENT
CMDLIN  DB      256 DUP(?)      ; COMMAND LINEの文字列を格納
DSRET   DD      1 DUP(?)        ; REAL MODEの割込処理から復帰した時点の
ESRET   DD      1 DUP(?)        ; DS, ES, FS, GSレジスタの値
FSRET   DD      1 DUP(?)        ; "
GSRET   DD      1 DUP(?)        ; "
```

メモリ管理用語集
スタック　一時的にデータを格納するための記憶領域．関数や手続き処理において引数やアドレスを格納するのに使われる．

のロードが完了すると，タスク0を通じてタスク2すなわちコマンドファイルのプログラムを起動する．このようにコマンドインタプリタは，CPUをリアルモードからプロテクトモードに切り替えてタスク0を起動する能力をもっている．

このほかコマンドインタプリタは，RコマンドとQコマンドをサポートしている．Rコマンドは，例外発生時，あるいはプログラム終了時のCPUの各レジスタの内容を表示する．Qコマンドは P386を終了させ，システムの制御を MS-DOSに戻す．なお，P386のコマンドラインは，P386起動時に MS-DOSのコマンドラインからも指定できる．MS-DOSのコマンドラインに，P386のコマンドラインが指定されたときには，コマンドラインによって指定されたプログラムが正常に起動され，正常に終了したときには，P386はただちに終了する．

タスク0

タスク0は，コマンドインタプリタがCPUのモードをプロテクトモードに切り替えると実行を開始する．タスク0自身の体裁を整えるため，簡単な初期化を行ったあと，タスク1またはタスク2のいずれかを起動する．起動するタスクの区別は，単にタスク0の

```
ERRSAV  DD      1 DUP(?)            ; ERROR CODEのある場合NON ZERO
ERRCOD  DD      1 DUP(?)            ; ERROR CODE
HINTNO  DD      1 DUP(?)            ; H/W割込のある場合、対応するVECTOR番号
FUNCNO  DD      1 DUP(?)            ; 割込、例外のVECTOR番号
CR0SAV  DD      1 DUP(?)            ; 割込、例外発生時のCPUの各REGISTER
CR2SAV  DD      1 DUP(?)            ; 〃
CR3SAV  DD      1 DUP(?)            ; 〃
EIPSAV  DD      1 DUP(?)            ; 〃
EFSAV   DD      1 DUP(?)            ; 〃
EAXSAV  DD      1 DUP(?)            ; 〃
EBXSAV  DD      1 DUP(?)            ; 〃
ECXSAV  DD      1 DUP(?)            ; 〃
EDXSAV  DD      1 DUP(?)            ; 〃
ESISAV  DD      1 DUP(?)            ; 〃
EDISAV  DD      1 DUP(?)            ; 〃
EBPSAV  DD      1 DUP(?)            ; 〃
ESPSAV  DD      1 DUP(?)            ; 〃
CSSAV   DD      1 DUP(?)            ; 〃
DSSAV   DD      1 DUP(?)            ; 〃
ESSAV   DD      1 DUP(?)            ; 〃
FSSAV   DD      1 DUP(?)            ; 〃
GSSAV   DD      1 DUP(?)            ; 〃
SSSAV   DD      1 DUP(?)            ; 〃
TRSAV   DD      1 DUP(?)            ; 〃
LDTSAV  DD      1 DUP(?)            ; 〃
GDTSAV  DQ      1 DUP(?)            ; 〃
IDTSAV  DQ      1 DUP(?)            ; 〃
COMEND  EQU     $
COM     ENDS
;
;       REAL MODEのDOS PROGRAMとPROTECTED MODEのTASK 0
;
TEXT    SEGMENT
        ASSUME  CS:TEXT,DS:DATA
;
;       本PROGRAM自体に必要な初期化
;
START:
        MOV     BX,SS                       ; 不要なメモリを解放
        MOV     DX,SP
        ADD     DX,15
        MOV     CL,4
        SHR     DX,CL
        ADD     BX,DX
        MOV     DX,DS
        SUB     BX,DX
        MOV     AH,4AH                      ; MEMORY割当量の変更
        INT     MSDOS
        CLD                                 ; コマンドラインのパラメータを
        MOV     AX,SEG DATA                 ; CBUFへ転送
        MOV     ES,AX
        MOV     SI,0080H
        MOV     DI,OFFSET CBUF
        MOV     CL,[SI]
        AND     CL,CL
        JZ      START1
        INC     SI
        MOV     CH,0
        REP     MOVSB
START1:
        MOV     AX,ES
        MOV     DS,AX
        MOV     BYTE PTR [DI],0
        SGDT    FWORD PTR GDTORG            ; BIOSで設定されているGDTRと
        SIDT    FWORD PTR IDTORG            ; IDTRをSAVE
;
;       BUF, LDT, STACK(PROTECTED MODE用)の各SEGMENTのための
;       MEMORYを確保
;
        MOV     AH,48H                      ; BUF SEGMENTのメモリを確保
        MOV     BX,1000H        ; 64KB
        INT     MSDOS
        JNC     START3
        JMP     MEMERR
START3:
        MOV     BUFTOP,AX
        MOV     AH,48H                      ; LDT SEGMENTのメモリを確保
        MOV     BX,200H         ; 8KB
        INT     MSDOS
        JNC     START4
        JMP     MEMERR
START4:
        MOV     LDTTOP,AX
        MOV     DI,OFFSET STK00             ; TASK 0,1,2のSTACK SEGMENT
        MOV     CX,12                       ; 領域を確保
START6:
        MOV     AH,48H
        MOV     BX,STKSIZ/16
        INT     MSDOS
        JNC     START5
        JMP     MEMERR
START5:
        STOSW
        LOOP    START6
;
;       共通に使用するDESCRIPTORの設定
;
        CLD                                 ; GDT SEGMENTのDESCRIPTOR
        MOV     DI,OFFSET GDTR              ; の初期化(LGDT用)
        MOV     AX,OFFSET GDTEND-1
        STOSW
        MOV     AX,SEG GDT
        XOR     DX,DX
        CALL    LINEAR
        STOSW
        MOV     AX,DX
        STOSW
        XOR     AX,AX
        STOSW
        MOV     DI,OFFSET IDTR              ; IDT SEGMENTのDESCRIPTOR
        MOV     AX,OFFSET IDTEND-1          ; の初期化(LIDT用)
        STOSW
        MOV     AX,SEG IDT
        XOR     DX,DX
        CALL    LINEAR
        STOSW
        MOV     AX,DX
        STOSW
        XOR     AX,AX
        STOSW
        MOV     AX,SEG GDT                  ; NULL DESCRIPTOR
        MOV     ES,AX
        MOV     DI,OFFSET NULL
        XOR     AX,AX
        MOV     CX,4
        CLD
        REP     STOSW
        MOV     AX,OFFSET GDTEND-1          ; GDTDESC DESCRIPTOR
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,SEG GDT
        MOV     DL,0
        MOV     DH,40H          ; GRANULE=BYTE, DEFAULT=32
        MOV     SI,0
        MOV     DI,OFFSET GDTDESC
        CALL    SETDSC
        MOV     AX,OFFSET IDTEND-1          ; IDTDESC DESCRIPTOR
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,SEG IDT
        MOV     DL,0
        MOV     DH,40H
        MOV     SI,0
        MOV     DI,OFFSET IDTDESC
        CALL    SETDSC
        MOV     AX,INTEND-INT00             ; INTDESC DESCRIPTOR
        MOV     BL,0            ; DPL=0
        MOV     BH,1AH          ; CODE、READ可能
        MOV     CX,SEG INTP     ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
```

メモリ管理用語集
スタンダードモード　Windows 3.0において80286以降のCPUで1MBを超えるメモリ空間を利用可能なモード．

エントリポイントで区別される．タスク1がIRET命令で終了するとタスク0に復帰するので，COMセグメントに終了コードを格納してCPUをリセットし，リアルモードに復帰する．

CPUをリセットする命令群は，CPURSTでマクロ化している．タスク2が終了した場合も同様であるが，唯一の違いは，デバッグに便利なように，タスク2のTSSセグメントから終了時のCPUのレジスタの内容をCOMセグメントにコピーすることである．

タスク1

TSSに書き込まれているパラメータをもとに，BUFセグメントからタスク2のコード領域へプログラムを転送する．転送終了後は，IRET命令でタスク0に復帰する．このタスクは誤動作を防止するため，INTR割り込みが禁止されている．

タスク2

コマンドファイルのプログラムを実行するタスクである．デフォルトのサイズは，32ビットに設定されている．タスク2起動時のインターフェースについてはあとで述べる．コード領域は110000Hから始まる64KBである．

```
        MOV     DI,OFFSET INTDESC
        CALL    SETDSC
        MOV     AX,2000H-1                  ; LDTDESC DESCRIPTOR
        MOV     BL,0            ; DPL=0
        MOV     BH,02H          ; LDT SYSTEM SEGMENT
        MOV     CX,LDTTOP       ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET LDTDESC
        CALL    SETDSC
        MOV     AX,OFFSET COMEND-1          ; COMDESC DESCRIPTOR
        MOV     BL,0            ; DPL=0
        MOV     BH,12H          ; DATA R/W
        MOV     CX,SEG COM      ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET COMDESC
        CALL    SETDSC
        MOV     AX,0FFFFH                   ; BUFDESC DESCRIPTOR
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,BUFTOP       ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET BUFDESC
        CALL    SETDSC
;
;       IDTに386割込GATEを256個設定
;
        MOV     AX,SEG IDT                  ; IDTの初期化
        MOV     ES,AX
        MOV     CX,256
        MOV     DI,OFFSET IDTBGN
        MOV     SI,0
START7:
        PUSH    CX
        PUSH    SI
        MOV     AX,SI           ; OFFSET下位
        MOV     BL,0            ; DPL=0
        MOV     BH,0EH          ; 386割込GATE
        MOV     CX,OFFSET INTDESC
        MOV     DX,0            ; OFFSET上位
        MOV     SI,0            ; DWORD COUNT
        CALL    SETGAT
        POP     SI
        POP     CX
        ADD     SI,8            ; ENTRY POINTは8バイト毎
        LOOP    START7
;
;       COM SEGMENTを0で初期化
;
        MOV     AX,SEG COM
        MOV     ES,AX
        XOR     AX,AX
        MOV     DI,OFFSET CMDLIN
        MOV     CX,COMEND-CMDLIN
        CLD
        REP     STOSB
        MOV     STAMOD,AX
;
;       単純なCOMMAND INTERPRETER
;
CNSL:
        MOV     AX,SEG DATA                 ; 起動時のENTRY POINT
        MOV     ES,AX
        XOR     AX,AX
        MOV     LDTRAP,AX
        MOV     SI,OFFSET CBUF
        CALL    SKPBLN
        AND     AL,AL
        JNZ     CNSL1
        XOR     AX,AX
        DEC     AX
        MOV     STAMOD,AX
CNSL0:
        MOV     AX,SEG DATA             ; PROTECTED MODEのPROGRAM終了時の
        MOV     DS,AX                   ; ENTRY POINT
        MOV     ES,AX
        MOV     AX,STAMOD               ; 本PROGRAM起動時のCOMMAND LINE
        AND     AX,AX                   ; でPROTECTED MODEのPROGRAMが
        JNZ     CNSL00                  ; 起動されていれば直ちに終了
        JMP     EXIT
CNSL00:
        MOV     SI,OFFSET MSG2          ; COMMAND PROMPT
        CALL    MSGOUT
        CALL    MSGIN                   ; COMMAND LINE入力
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        JMP     SHORT CNSL
CNSL1:
        PUSH    SI                      ; COMMAND LINEの解析
        MOV     DI,OFFSET CMD0          ; Q COMMAND?
        CALL    KYWD
        JNZ     CNSL2                   ; Q COMMANDではない
        CALL    SKPBLN
        AND     AL,AL
        JNZ     CNSL3                   ; Qで始まるFILE名?
        POP     SI
        JMP     EXIT
CNSL2:
        POP     SI
        PUSH    SI
        MOV     DI,OFFSET CMD1          ; R COMMAND?
        CALL    KYWD
        JNZ     CNSL3                   ; R COMMANDではない
        CALL    SKPBLN
        AND     AL,AL
        JNZ     CNSL3                   ; Rで始まるFILE名?
        POP     SI
        CALL    COMDSP
        JMP     CNSL0
CNSL3:
        POP     SI
        MOV     DI,OFFSET CMDFN         ; COMMAND FILE名?
        CALL    FNSET
        MOV     AL,CMDFN
        AND     AL,AL
        JNZ     CNSL4
        MOV     SI,OFFSET MSG3          ; 有効なFILE名でない
        CALL    MSGOUT
        JMP     CNSL0
CNSL4:
        MOV     AL,[SI]                 ; FILE拡張子あり?
        CMP     AL,'.'
        JNZ     CNSL5
        CALL    FTSET                   ; 拡張子をCOMMAND FILE名にSET
        JMP     SHORT CNSL6
CNSL5:
        PUSH    SI                      ; FILE拡張子なし
        MOV     SI,OFFSET FTDFLT        ; .BINをFILE拡張子とする
        MOV     CX,5
        CLD
        REP     MOVSB
        POP     SI
CNSL6:
        CALL    SKPBLN                  ; COMMAND LINEのPARAMETER部を
        MOV     DI,OFFSET CMDLIN        ; COM SEGMENTに転送
        MOV     AX,SEG COM
        MOV     ES,AX
        CLD
CNSL7:
        LODSB
        STOSB
        AND     AL,AL
        JNZ     CNSL7
        MOV     AH,3DH                  ; COMMAND FILEをOPEN
        MOV     AL,0
        MOV     DX,OFFSET CMDFN
        INT     MSDOS
```

メモリ管理用語集
スワップ（Swap）　メインメモリと拡張メモリ，メインメモリとディスクなどの間でデータやプログラムの交換をすること．

プロテクトモード側の割り込み処理

初期化ブロックによって，IDTに設定されている256個のエントリポイントをもつ割り込み処理プログラムである．各タスクからは，386割り込みゲートによって，このプログラムに制御が移動する．したがって，この処理は割り込まれたタスクの一部として実行される．各エントリポイントは，リスト1のマクロ命令IENTRYにあるように，割り込み番号をALレジスタに置いて共通の処理に移行する．共通の処理では，割り込みの原因を例外，INTR割り込み，およびソフトウェア割り込みに分類したうえで，エラーコードがあればここでスタックからポップする．INTR割り込みであれば，8259PICのマスタとスレーブから割り込み番号を求める．そして，これらの情報と割り込まれた時点でのCPUのレジスタの情報をCOMセグメントに書き込んで，CPUをリセットしリアルモードに復帰する．P386では，DRnとTRnの情報はサポートしていない．

リアルモード側割り込み処理

リスト1のマクロ命令RETRSTを実行しておくと，CPUリセット後

```
          JNC     CNSL8
          MOV     SI,OFFSET MSG4          ; COMMAND FILEがOPENできない
          CALL    MSGOUT
          MOV     SI,OFFSET CMDFN
          CALL    MSGOUT
          MOV     SI,OFFSET MSG5
          CALL    MSGOUT
          JMP     CNSL0
;
;         COMMAND FILEのLOAD
;
CNSL8:
          PUSH    AX
          MOV     AX,SEG GDT              ; DSTDESCをCOMMAND FILE
          MOV     ES,AX                   ; 転送先に初期化
          MOV     AX,0FFFFH
          MOV     BL,0            ; DPL=0
          MOV     BH,12H          ; DATA R/W
          MOV     CX,LDLO
          MOV     DL,0            ; LIMIT上位
          MOV     DH,40H          ; GRANUL=BYTE,DEFAULT=32
          MOV     SI,LDHI
          MOV     DI,OFFSET DSTDESC
          CALL    SETDSC
          POP     AX
          MOV     CMDHND,AX               ; FILE HANDLEをSAVE
          XOR     AX,AX                   ; EOF MARKをRESET
          MOV     CMDEOF,AX
CNSL13:
          XOR     AX,AX                   ; LOADすべきBYTE数 = 0
          MOV     WORD PTR CMDBYT,AX
          MOV     WORD PTR CMDBYT+2,AX
          DEC     AX                      ; LOAD中のFLAGをSET
          MOV     LDTRAP,AX
          MOV     AH,3FH                  ; COMMAND FILEをREAD
          MOV     BX,CMDHND
          MOV     CX,8000H        ; 32KB
          MOV     DX,0000H        ; BUF:0000Hへ
          PUSH    DS
          MOV     DS,BUFTOP
          INT     MSDOS
          POP     DS
          JNC     CNSL9
CNSL11:
          MOV     SI,OFFSET MSG4          ; COMMAND FILEがREADできない
          CALL    MSGOUT
          MOV     SI,OFFSET CMDFN
          CALL    MSGOUT
          MOV     SI,OFFSET MSG6
          CALL    MSGOUT
          MOV     AH,3EH
          MOV     BX,CMDHND
          INT     MSDOS
          XOR     AX,AX
          MOV     LDTRAP,AX               ; LOAD中のFLAGをRESET
          JMP     CNSL0
CNSL9:
          ADD     WORD PTR CMDBYT,AX      ; 実際に読み込まれたBYTE数を
          ADC     WORD PTR CMDBYT+2,0     ; LOADすべきBYTE数に加算
          CMP     AX,8000H
          JNZ     CNSL10
          MOV     AH,3FH                  ; COMMAND FILEのREAD
          MOV     BX,CMDHND
          MOV     CX,8000H        ; 32KB
          MOV     DX,8000H        ; BUF:8000Hへ
          PUSH    DS
          MOV     DS,BUFTOP
          INT     MSDOS
          POP     DS
          JC      CNSL11
          ADD     WORD PTR CMDBYT,AX      ; 実際に読み込まれたBYTE数を
          ADC     WORD PTR CMDBYT+2,0     ; LOADすべきBYTE数に加算
          CMP     AX,8000H
          JZ      CNSL12
CNSL10:
          MOV     AH,3EH                  ; COMMAND FILEがEOFになった
          MOV     BX,CMDHND
          INT     MSDOS
          XOR     AX,AX
          DEC     AX
          MOV     CMDEOF,AX                       ; EOFをMARK
CNSL12:
;
          CALL    SETUP0                          ; TASK 0を初期化
          CALL    SETUP1                          ; TASK 1を初期化
;
          MOV     CX,WORD PTR CMDBYT
          MOV     DX,WORD PTR CMDBYT+2
          ASSUME  DS:TS1TSS
          MOV     AX,SEG TS1TSS
          MOV     DS,AX
          MOV     WORD PTR TS1ECX,CX
          MOV     WORD PTR TS1ECX+2,DX
          ASSUME  DS:DATA
          MOV     AX,SEG DATA                     ; COMMAND FILEの転送元を
          MOV     DS,AX                           ; SRCDESCにSET
          MOV     AX,SEG GDT
          MOV     ES,AX
          MOV     AX,0FFFFH       ; LIMIT=64KB
          MOV     BL,0            ; DPL=0
          MOV     BH,12H          ; DATA R/W
          MOV     CX,BUFTOP       ; BASE下位
          MOV     DL,0            ; LIMIT上位
          MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
          MOV     SI,0            ; BASE上位
          MOV     DI,OFFSET SRCDESC
          CALL    SETDSC
;
          RETRST                                  ; RESET後の処理継続指定
          PUSHF                                   ; IOPL=0, NT=0
          POP     AX
          AND     AX,0FFFH
          PUSH    AX
          POPF
          CLI
          MOV     AX,SEG DATA
          MOV     DS,AX
          LGDT    FWORD PTR GDTR          ; GDTRの初期化
          LIDT    FWORD PTR IDTR          ; IDTRの初期化
          MOV     EAX,CR0                 ; CR0のPEをSET
          OR      AL,01H
          MOV     CR0,EAX
;
;         ここからPROTECTED MODE TASK 0 (TASK 1の起動)
;
          JMP     SHORT $+2                       ; PREFETCH QUEUEをFLUSH
          XOR     AX,AX
          LLDT    AX
          MOV     AX,OFFSET ST0DESC               ; STACKの初期化
          MOV     SS,AX
          MOV     SP,STKSIZ
          MOV     AX,OFFSET TS0DESC               ; TRの初期化
          LTR     AX
          MOV     AX,OFFSET NULL                  ; DS, ES, FS, GSの初期化
          MOV     DS,AX
          MOV     ES,AX
          MOV     FS,AX
          MOV     GS,AX
          JMPF    NXT0,TS0CODE                    ; CSの初期化
NXT0:
          CALLF   0,TS1DESC                       ; TASK 1の呼出
          MOV     AX,OFFSET COMDESC
          MOV     DS,AX
          ASSUME  DS:COM
          MOV     AX,0100H                        ; TASK1の終了コード
          MOV     WORD PTR FUNCNO,AX
          CPURST                                  ; CPUのRESET
;
;
;         ここからREAL MODE (PROGRAMのLOAD継続とRUN)
;
          ASSUME  DS:DATA
```


メモリ管理用語集
静的システム　ソフトウェアのサイズが実装メモリよりも小さく，すべてをメモリ上にロードできるシステム．

の処理継続アドレスをラベルのRESETとすることができる．これを利用しておくと，CPUがリセットされたとき，つねにリアルモード側の割り込み処理へ制御を移すことができる．

リアルモード側の割り込み処理では，COMセグメントによってプロテクトモード側が設定した情報を使用して，再び割り込みの原因をタスク終了，例外，INTR割り込み，およびソフトウェア割り込みに分類する．

タスク終了の場合は，直接コマンドインタプリタに戻る．

例外の場合は，例外の種別とCOMセグメントにセットされているCPUのレジスタの値を表示して，コマンドインタプリタに戻る．

INTR割り込みの場合には，割り込み番号に対応するINT n命令を実行する．そしてリアルモードからプロテクトモードに切り替え，全レジスタを割り込み時の状態に復旧して，IRET命令で直接割り込まれたタスクに復帰する．

ソフトウェア割り込みの場合には，INTR割り込みと異なり割り込み処理先と割り込んだタスクとの間でパラメータの授受が必要になる．そこで，最初にSPを除く汎用レジスタのすべてとFLAGSレジスタをCOMセグメントから復旧させる．DS~GSのセレクタは，タスク側の値を渡しても無意味であるので，すべてBUFセグメントを指すように設定する．そしてINT n命令を実行してリアルモードの割り込みサービスを受ける．サービ

```
CNSLX:
        MOV     AX,CMDEOF
        AND     AX,AX
        JNZ     CNSLX1                  ; FILEをEOFまでLOADした
        MOV     AX,SEG GDT              ; FILEを更にLOAD
        MOV     ES,AX
        MOV     DI,OFFSET DSTDESC
        INC     BYTE PTR ES:4[DI]       ; DSTDESCのBASE ADDRESS + 64KB
        JMP     CNSL13
;
;       COMMAND FILEのPROGRAMをTASK 2として実行させる
;
CNSLX1:
        XOR     AX,AX
        MOV     LDTRAP,AX               ; LOAD中FLAGをOFF
        CALL    SETUP0                  ; TASK 0の初期化
        CALL    SETUP2                  ; TASK 2の初期化
        RETRST                          ; RESET後の処理継続指定
        PUSHF                           ; IOPL=0, NT=0
        POP     AX
        AND     AX,0FFFH
        PUSHF
        CLI                             ; TASK 0実行中は割込禁止
        MOV     AX,SEG DATA
        MOV     DS,AX
        LGDT    FWORD PTR GDTR          ; GDTRの初期化
        LIDT    FWORD PTR IDTR          ; IDTRの初期化
        MOV     EAX,CR0                 ; CR0のPEをSET
        OR      AL,01H
        MOV     CR0,EAX
;
;       ここからPROTECTED MODE TASK 0 (TASK 2の起動)
;
        JMP     SHORT $+2               ; PREFETCH QUEUEのFLUSH
        XOR     AX,AX                   ; LDTRをNULL SELECTORで初期化
        LLDT    AX
        MOV     AX,OFFSET ST0DESC       ; STACKの初期化
        MOV     SS,AX
        MOV     SP,STKSIZ
        MOV     AX,OFFSET TS0DESC       ; TRの初期化
        LTR     AX
        MOV     AX,OFFSET NULL          ; DS, ES, FS, GSの初期化
        MOV     DS,AX
        MOV     ES,AX
        MOV     FS,AX
        MOV     GS,AX
        JMPF    NXT1,TS0CODE            ; CSの初期化
NXT1:
        CALLF   0,TS2DESC               ; TASK 2の呼出(実行開始)
        MOV     AX,OFFSET GDTDESC       ; TS2DESCをSRCDESCへCOPYして
        MOV     DS,AX                   ; TS2TSS SEGMENTをDATA ACCESS
        MOV     ES,AX                   ; 可能にする
        ASSUME  DS:GDT
        MOV     SI,OFFSET TS2DESC
        MOV     DI,OFFSET SRCDESC
        CLD
        MOVSW
        MOVSW
        MOVSW
        MOVSW
        MOV     AX,0092H        ; DATA R/W,GRANULE=BYTE,DEFALUT=16
        MOV     WORD PTR SRCDESC+5,AX
        MOV     AX,OFFSET SRCDESC
        MOV     DS,AX
        MOV     AX,OFFSET COMDESC
        MOV     ES,AX
        ASSUME  DS:TS2TSS,ES:COM
        MOV     ESI,OFFSET TS2CR3       ; TS2TSS SEGMENTの内容を
        LODSD                           ; COM SEGMENTにCOPYする
        MOV     ES:CR3SAV,EAX
        LODSD
        MOV     ES:EIPSAV,EAX
        LODSD
        MOV     ES:EFSAV,EAX
        LODSD
        MOV     ES:EAXSAV,EAX
```

```
        LODSD
        MOV     ES:ECXSAV,EAX
        LODSD
        MOV     ES:EDXSAV,EAX
        LODSD
        MOV     ES:EBXSAV,EAX
        LODSD
        MOV     ES:ESPSAV,EAX
        LODSD
        MOV     ES:EBPSAV,EAX
        LODSD
        MOV     ES:ESISAV,EAX
        LODSD
        MOV     ES:EDISAV,EAX
        LODSD
        MOV     ES:ESSAV,EAX
        LODSD
        MOV     ES:CSSAV,EAX
        LODSD
        MOV     ES:SSSAV,EAX
        LODSD
        MOV     ES:DSSAV,EAX
        LODSD
        MOV     ES:FSSAV,EAX
        LODSD
        MOV     ES:GSSAV,EAX
        LODSD
        MOV     ES:LDTSAV,EAX
        SGDT    FWORD PTR ES:GDTSAV     ; GDTRをCOPY
        SIDT    FWORD PTR ES:IDTSAV     ; IDTRをCOPY
        MOV     AX,OFFSET COMDESC
        MOV     DS,AX
        ASSUME  DS:COM,ES:NOTHING
        MOV     AX,0101H                ; RESET原因CODEをSET
        MOV     WORD PTR FUNCNO,AX
        CPURST                          ; CPUのRESET
;
;
;       ここからREAL MODE (CPU RESETの原因解析)
;
RESET:
        MOV     AX,SEG STACK            ; REAL MODEのSTACKを初期化
        MOV     SS,AX
        MOV     SP,400H
        RETRST                          ; RESET後の処理継続指定
        MOV     AX,SEG COM
        MOV     DS,AX
        MOV     AX,WORD PTR FUNCNO      ; RESET原因CODEを調べる
        CMP     AX,0100H
        JC      RESET0                  ; 割込あるいは例外が発生している
        ADD     SP,4                    ; TASK 1またはTASK 2が終了した
        MOV     AX,SEG DATA
        MOV     DS,AX
        MOV     ES,AX
        ASSUME  DS:DATA
        LGDT    FWORD PTR GDTORG        ; GDTRを初期化
        LIDT    FWORD PTR IDTORG        ; IDTRを初期化
        STI                             ; 割込を許可
        MOV     AX,LDTRAP
        AND     AX,AX
        JNZ     RESET8                  ; COMMAND FILEをLOAD中
        JMP     CNSL0                   ; PROTECTED MODE PROGRAMの終了
RESET8:
        JMP     CNSLX
RESET0:
        CMP     AL,8
        JC      RESET1
        JMP     RESET2
RESET1:
        MOV     SI,OFFSET MSGT0         ; 例外0から7の場合
        CMP     AL,0
        JZ      RESET3
        MOV     SI,OFFSET MSGT1
        CMP     AL,1
        JZ      RESET3
        MOV     SI,OFFSET MSGT2
```

メモリ管理用語集

セグメント　8086の実アドレスはオフセットとセグメントを4ビット左にシフトさせた値との和で決定される．1MBのメモリ空間を16ビットで操作するために導入された．

ス終了後は，今度は逆にSPを除く汎用レジスタのすべてとFLAGSをCOMセグメントにセーブする．割り込みサービスから戻されたDS~GSの値は，タスクに直接渡せないのでCOMセグメントの特別な領域に保存する．ここで，リアルモードからプロテクトモードに移行して，COMセグメントからCPUのレジスタを復旧し，IRET命令で直接割り込んだタスクに復帰する．

割り込みと例外の判定

PC-9801シリーズでは，表3に示すようにINTR割り込みと例外が同じ割り込み番号に割り当てられている．INTR割り込みと例外が同時に近いタイミングで発生したときには，100%完全に分離することはできないが，このプログラムでは次のような方法でINTR割り込み，ソフトウェア割り込み，例外の判定をしている．

①割り込み番号が0～7までの場合は例外と判定する．
②割り込み番号が8～17までの場合は，PICのマスタとスレーブを読んで，INTR割り込みの有無と割り込みのある場合，対応する割り込み

```
        CMP     AL,2
        JZ      RESET3
        MOV     SI,OFFSET MSGT3
        CMP     AL,3
        JZ      RESET3
        MOV     SI,OFFSET MSGT4
        CMP     AL,4
        JZ      RESET3
        MOV     SI,OFFSET MSGT5
        CMP     AL,5
        JZ      RESET3
        MOV     SI,OFFSET MSGT6
        CMP     AL,6
        JZ      RESET3
        MOV     SI,OFFSET MSGT7
RESET3:
        MOV     AX,SEG DATA
        MOV     DS,AX
        MOV     ES,AX
        LGDT    FWORD PTR GDTORG        ; GDTRの初期化
        LIDT    FWORD PTR IDTORG        ; IDTRの初期化
        STI                             ; 割込を許可
        CALL    MSGOUT                  ; 例外の原因を表示
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        CALL    COMDSP                  ; 例外発生時のCPUの状態を表示
        ADD     SP,4
        JMP     CNSL0                   ; 実行を終了
        ASSUME  DS:COM
RESET2:
        CMP     AL,18H
        JC      RESET4
        JMP     RESET5
RESET4:
        MOV     BX,WORD PTR HINTNO      ; H/W割込あり?
        AND     BX,BX
        JS      RESET6                  ; YES
RESET7:
        MOV     SI,OFFSET MSGT8         ; 例外8から16の場合(H/W割込なし)
        CMP     AL,8
        JZ      RESET3
        MOV     SI,OFFSET MSGT9
        CMP     AL,9
        JZ      RESET3
        MOV     SI,OFFSET MSGT10
        CMP     AL,10
        JZ      RESET3
        MOV     SI,OFFSET MSGT11
        CMP     AL,11
        JZ      RESET3
        MOV     SI,OFFSET MSGT12
        CMP     AL,12
        JZ      RESET3
        MOV     SI,OFFSET MSGT13
        CMP     AL,13
        JZ      RESET3
        MOV     SI,OFFSET MSGT14
        CMP     AL,14
        JZ      RESET3
        MOV     SI,OFFSET MSGT16
        CMP     AL,16
        JZ      RESET3                  ; MESSAGEを表示して処理終了
        JMP     RESET5                  ; S/W割込の処理を行う
RESET6:
        CMP     AL,BL
        JNZ     RESET7                  ; 割込と対応するVECTOR番号が相違
;
;       H/W割込処理を呼び出す
;
        MOV     CS:CDXX+1,AL            ; VECTOR番号をINT命令に組み込む
        MOV     AX,SEG DATA
        MOV     DS,AX
        ASSUME  DS:DATA
        LGDT    FWORD PTR GDTORG        ; GDTRをORIGINALに戻す
        LIDT    FWORD PTR IDTORG        ; IDTRをORIGNALに戻す
CDXX    DB      0CDH,00H                ; INT xx命令
        CLI                             ; 割込処理で何が実行されるか
        PUSHF                           ; 不明なので念のため
        POP     AX                      ; 〃
        AND     AX,0FFFH                ; 〃 IOPL=0, NT=0
        PUSH    AX                      ; 〃
        POPF                            ; 〃
;
;       H/W割込なのでCPUの状態を変更しない
;
        MOV     AX,SEG COM
        MOV     DS,AX
        ASSUME  DS:COM
        LGDT    FWORD PTR GDTSAV        ; GDTRを復旧
        LIDT    FWORD PTR IDTSAV        ; IDTRを復旧
        MOV     EAX,CR0SAV              ; CR0を復旧
        MOV     CR0,EAX
;
;       ここからPROTECTED MODE, TASK 2
;
        JMPF    NXT2,TS0CODE            ; PREFETCH QUEUEをFLUSH
NXT2:
        MOV     AX,OFFSET COMDESC
        MOV     DS,AX
        MOV     SS,WORD PTR SSSAV       ; SS:SPを復旧
        MOV     ESP,ESPSAV
        LLDT    WORD PTR LDTSAV         ; LDTRを復旧
        LTR     WORD PTR TRSAV          ; TRを復旧
        MOV     EAX,CR3SAV              ; CR3を復旧
        MOV     CR3,EAX
        MOV     AX,WORD PTR DSSAV       ; DSの復旧準備
        PUSH    AX
        MOV     EAX,EAXSAV              ; 各REGISTERを復旧
        MOV     EBX,EBXSAV              ; 〃
        MOV     ECX,ECXSAV              ; 〃
        MOV     EDX,EDXSAV              ; 〃
        MOV     ESI,ESISAV              ; 〃
        MOV     EDI,EDISAV              ; 〃
        MOV     EBP,EBPSAV              ; 〃
        MOV     ES,WORD PTR ESSAV       ; 〃
        MOV     FS,WORD PTR FSSAV       ; 〃
        MOV     GS,WORD PTR GSSAV       ; 〃
        POP     DS                      ; 〃
        IRETD                           ; 割込元TASKにRETURN
;
;       ここからREAL MODE(S/W割込の場合の処理)
;
RESET5:
        MOV     CS:CDYY+1,AL            ; INT xx命令にVECTOR番号を組込む
        MOV     AX,SEG DATA
        MOV     DS,AX
        ASSUME  DS:DATA
        MOV     AX,BUFTOP               ; 各SEGMENT SELECTORは
        PUSH    AX                      ; BUF SEGMENTを指定
        PUSH    AX
        PUSH    AX
        PUSH    AX
        LGDT    FWORD PTR GDTORG        ; GDTRをORIGINALに戻す
        LIDT    FWORD PTR IDTORG        ; IDTRをORIGINALに戻す
        MOV     AX,SEG COM              ; 割込処理に渡すPARAMETERを準備
        MOV     DS,AX
        ASSUME  DS:COM
        MOV     AX,WORD PTR EFSAV
        AND     AX,0FFFH
        PUSH    AX
        MOV     AX,WORD PTR EAXSAV
        MOV     BX,WORD PTR EBXSAV
        MOV     CX,WORD PTR ECXSAV
        MOV     DX,WORD PTR EDXSAV
        MOV     SI,WORD PTR ESISAV
        MOV     DI,WORD PTR EDISAV
        MOV     BP,WORD PTR EBPSAV
        POPF
        POP     GS
        POP     FS
        POP     ES
        POP     DS
```

メモリ管理用語集
セレクタ　プロテクトモード時セグメントレジスタはセレクタとして利用され，ディスクリプタテーブルへのインデックスとして利用される．

番号を調べる．割り込みがあり，割り込み番号とPICから求めた割り込み番号が一致する場合には，INTR割り込みとみなす．これ以外の場合には，割り込み番号に例外が割り当てられている場合は例外，そうでない場合はソフトウェア割り込みとみなす．

③割り込み番号が18H以上の場合には，無条件にソフトウェア割り込みと判定する．

リアルモードとプロテクトモードの移行

386でリアルモードからプロテクトモードへ移行するには，次の手順を踏

表3：INTR割り込みと例外の重複

割り込み番号	INTR 割り込み	例　外
08	タイマ	ダブルフォールト
09	キーボード	NDP セグメント
0A	CRT VSYNC	不当 TSS
0B	拡張バス INT0	不在セグメント
0C	RS-232C	不当 SS
0D	拡張バス INT1	一般保護
0E	拡張バス INT2	ページフォールト
0F	PIC スレーブ	
10		NDP エラー
11	拡張バス INT3(HD)	不適切な境界整列
12	拡張バス INT41(640KB FD)	
13	拡張バス INT42(1MB FD)	
14	拡張バス INT5	
15	拡張バス INT6(マウス)	
16		
15		

```
CDYY    DB      0CDH,00H                ; INT xx命令
        CLI
;
;       S/W割込の場合、汎用REGISTORとFLAGSを割込処理終了時点の
;       値に変更して、割込元に返す
;
        PUSH    DS
        PUSH    AX
        MOV     AX,SEG COM              ; 割込元にPARAMETERを戻す
        MOV     DS,AX
        POP     AX
        MOV     WORD PTR EAXSAV,AX      ; 汎用REGISTERの上位16 BITSは
        MOV     WORD PTR EBXSAV,BX      ; 不変
        MOV     WORD PTR ECXSAV,CX
        MOV     WORD PTR EDXSAV,DX
        MOV     WORD PTR ESISAV,SI
        MOV     WORD PTR EDISAV,DI
        MOV     WORD PTR EBPSAV,BP
        POP     AX
        MOV     WORD PTR DSRET,AX       ; DATA SEGMENTのSELECTORは
        MOV     WORD PTR ESRET,ES       ; 割込元へ返さない
        MOV     WORD PTR FSRET,FS
        MOV     WORD PTR GSRET,GS
        PUSHF
        POP     AX
        AND     AX,0FFFH                ; FLAGSの上位4 BITSは変更しない
        PUSH    AX                      ; EFLAGSの上位16 BITSは不変
        POPF
        MOV     BX,WORD PTR EFSAV
        AND     BX,0F000H
        OR      AX,BX
        MOV     WORD PTR EFSAV,AX
        LGDT    FWORD PTR GDTSAV        ; GDTRを復旧
        LIDT    FWORD PTR IDTSAV        ; IDTRを復旧
        MOV     EAX,CR0SAV              ; CR0を復旧
        MOV     CR0,EAX
;
;       ここからPROTECTED MODE, TASK 1,2
;
        JMPF    NXT3,TS0CODE            ; PREFETCH QUEUEをFLUSH
NXT3:
        MOV     AX,OFFSET COMDESC
        MOV     DS,AX
        MOV     SS,SSSAV                ; SS:ESPを復旧
        MOV     ESP,ESPSAV
        LLDT    WORD PTR LDTSAV         ; LDTRを復旧
        LTR     WORD PTR TRSAV          ; TRを復旧
        MOV     EAX,CR3SAV              ; CR3を復旧
        MOV     CR3,EAX
        MOV     AX,WORD PTR DSSAV       ; DSの復旧準備
        PUSH    AX
        MOV     EAX,EFSAV                       ; EFLAGSを返す
        MOV     EBP,ESP
        MOV     10[EBP],EAX
        MOV     ESI,ESISAV                      ; 各REGISTERを返す
        MOV     EDI,EDISAV                      ; "
        MOV     EAX,EAXSAV                      ; "
        MOV     EBX,EBXSAV                      ; "
        MOV     ECX,ECXSAV                      ; "
        MOV     EDX,EDXSAV                      ; "
        MOV     EBP,EBPSAV                      ; "
        MOV     ES,ESSAV                        ; SEGMENT SELECTORを復旧
        MOV     FS,FSSAV                        ; "
        MOV     GS,GSSAV                        ; "
        POP     DS                              ; "
        IRETD                                   ; 割込元TASKに復旧
        ASSUME  DS:DATA
;
;       ここからREAL MODE
;
EXIT:
        MOV     AH,4CH                          ; 本PROGRAMの終了
        MOV     AL,0
        INT     MSDOS
MEMERR:
        MOV     SI,OFFSET MSG1
        CALL    MSGOUT
        JMP     SHORT EXIT
;
; TASK 0関係のDESCRIPTORとTSSの初期設定
;
SETUP0:
        PUSH    DS
        PUSH    ES
        MOV     AX,SEG DATA
        MOV     DS,AX
        MOV     AX,SEG GDT
        MOV     ES,AX
        MOV     AX,OFFSET TSK0END-1     ; TASK 0 CODE
        MOV     BL,0            ; DPL
        MOV     BH,1AH          ; CODE READ可能
        MOV     CX,SEG TEXT     ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,0            ; GRANULE=BYTE,DEFAULT=16
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS0CODE
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1                     ; 特権LEVEL 0用のSTACK
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK00        ; BASE下位
```

メモリ管理用語集

増設メモリ　1MBを超える空間にマッピングできるメモリのこと．プロテクトメモリとも呼ばれ，プロテクトモードや仮想86モードからリニアにアクセスできる．

む．不要なトラブルを避けるため割り込みは禁止しておいたほうが無難である．

①LGDTとLIDT命令を実行してGDTRとIDTRを初期化する．
②MOVあるいはLMSW命令を実行してCR0のPEビットを1にする．
③FAR JMP命令を実行して適当な位置にジャンプし，プリフェッチキューをフラッシュすると同時にCSを初期化する．
④LLDTとLTR命令を実行してLDTRとTRを初期化する．
⑤CS以外のセグメントレジスタに有効なセレクタをロードする．
⑥ESPを初期化する．

386では，2つの方法でプロテクトモードからリアルモードへ移行できる．第1の方法は，286と同様にCPUをリセットする方法である．第2の方法は，MOV命令を実行してCR0のPEビットを0にする方法である．以下に示すのは第2の方法である．

①セグメントサイズがちょうど64KBのセグメントへFAR JMPを実行する．
②CSを除くすべてのセグメントレジスタに，セグメントサイズがちょうど64KBの合法なセレクタをロードする．
③MOV命令を実行してCR0のPEビットを0とする．
④リアルモードで有効なアドレスへFAR JMPして，プリフェッチキューをフラッシュすると同時にCSを

```
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS0STK0
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 特権LEVEL 1用のSTACK
        MOV     BL,1            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK01        ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS0STK1
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 特権LEVEL 2用のSTACK
        MOV     BL,2            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK02        ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS0STK2
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 初期使用のSTACK
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK0TOP      ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET ST0DESC
        CALL    SETDSC
;
        MOV     AX,OFFSET TS0END-1      ; TSS DESCRIPTOR
        MOV     BL,0            ; DPL
        MOV     BH,09H          ; 使用可能なTSS
        MOV     CX,SEG TS0TSS   ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,0            ; GRANULE=BYTE
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS0DESC
        CALL    SETDSC
;
        MOV     AX,SEG TS0TSS           ; TSS SEGMENTの初期設定
        MOV     ES,AX
        XOR     AX,AX
        MOV     CX,OFFSET TS0END
        SHR     CX,1
        MOV     DI,0
        CLD
        REP     STOSW
        MOV     AX,ES
        MOV     DS,AX
        ASSUME  DS:TS0TSS
        MOV     AX,STKSIZ               ; SPの初期値
        MOV     WORD PTR TS0ESP0,AX
        MOV     WORD PTR TS0ESP1,AX
        MOV     WORD PTR TS0ESP2,AX
        MOV     WORD PTR TS0ESP,AX
        MOV     AX,0202H                ; 割込許可、IOPL=0
        MOV     WORD PTR TS0EF,AX
        MOV     AX,OFFSET TS0STK0
        MOV     WORD PTR TS0SS0,AX
        MOV     AX,OFFSET TS0STK1
        MOV     WORD PTR TS0SS1,AX
        MOV     AX,OFFSET TS0STK2
        MOV     WORD PTR TS0SS2,AX
        MOV     AX,OFFSET ST0DESC
        MOV     WORD PTR TS0SS,AX
        MOV     AX,OFFSET TS0CODE
        MOV     WORD PTR TS0CS,AX
        MOV     AX,0068H                ; I/O BIT MAP OFFSET
        MOV     TS0IOM,AX
        POP     ES
        POP     DS
        RET
        ASSUME  DS:DATA
;
;       TASK 1関係のDESCRIPTORとTSSの初期化
;
SETUP1:
        PUSH    DS
        PUSH    ES
        MOV     AX,SEG DATA
        MOV     DS,AX
        MOV     AX,SEG GDT
        MOV     ES,AX
        MOV     EAX,OFFSET TSK1END-1    ; TASK 1 CODE
        MOV     BL,0            ; DPL
        MOV     BH,1AH          ; CODE READ可能
        MOV     CX,SEG TASK1    ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS1CODE
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 特権LEVEL 0用のSTACK
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK10        ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS1STK0
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 特権LEVEL 1用のSTACK
        MOV     BL,1            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK11        ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS1STK1
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 特権LEVEL 2用のSTACK
        MOV     BL,2            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK12        ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS1STK2
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1             ; 初期使用のSTACK
        MOV     BL,0            ; DPL
        MOV     BH,12H          ; DATA R/W
        MOV     CX,STK1TOP      ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,40H          ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET ST1DESC
        CALL    SETDSC
;
        MOV     AX,OFFSET TS1END-1      ; TSS DESCRIPTOR
        MOV     BL,0            ; DPL
        MOV     BH,09H          ; 使用可能な386TSS
        MOV     CX,SEG TS1TSS   ; BASE下位
        MOV     DL,0            ; LIMIT上位
        MOV     DH,0            ; GRANULE=BYTE
        MOV     SI,0            ; BASE上位
        MOV     DI,OFFSET TS1DESC
        CALL    SETDSC
;
        MOV     AX,SEG TS1TSS           ; TSS SEGMENTの初期設定
        MOV     ES,AX
```

メモリ管理用語集
タイムシェアリング　マルチタスクシステムにおいて，タイマを利用して一定時間ごとにタスクを切り替えること．

初期化する．

⑤LIDT命令を実行してリアルモード用にIDTRを初期化する．

タスク2の起動と終了，サンプルプログラム

タスク2の起動時に初期化されるレジスタは，SETUP2のサブルーチンに記述してあるとおりである．DSとESには，CSと同じベースアドレスからはじまる1MBの大きさのR/W可能なデータセグメントをアクセスできるセレクタで初期化される．FSには，COMセグメントのセレクタがロードされる．COMセグメントには，COMセグメントの先頭から00Hで終わるコマンドラインで指定されたパラメータの文字列が入っている．GSにはBUFセグメントのセレクタがロードされる．タスク2の特権レベルは0，IOPLも0である．割り込みは許可されており，4KBのスタックが使用できる状態になっている．

タスク2は0番地より実行を開始する．したがって，プログラムも0番地からはじまるように書かなければならない．オペランドサイズとアドレスサイズのデフォルトは32ビットにセットされるので，コードセグメントはUSE32である必要がある．

タスク2を終了させるには，IRET

```
        XOR     AX,AX
        MOV     CX,OFFSET TS1END
        SHR     CX,1
        MOV     DI,0
        CLD
        REP     STOSW
        MOV     AX,ES
        MOV     DS,AX
        ASSUME  DS:TS1TSS
        MOV     AX,STKSIZ                       ; SPの初期値
        MOV     WORD PTR TS1ESP0,AX
        MOV     WORD PTR TS1ESP1,AX
        MOV     WORD PTR TS1ESP2,AX
        MOV     WORD PTR TS1ESP,AX
        MOV     AX,0202H                        ; 割込許可、IOPL=0
        MOV     WORD PTR TS1EF,AX
        MOV     AX,OFFSET TS1STK0               ; 各SELECTORの初期値
        MOV     WORD PTR TS1SS0,AX
        MOV     AX,OFFSET TS1STK1
        MOV     WORD PTR TS1SS1,AX
        MOV     AX,OFFSET TS1STK2
        MOV     WORD PTR TS1SS2,AX
        MOV     AX,OFFSET ST1DESC
        MOV     WORD PTR TS1SS,AX
        MOV     AX,OFFSET LDTDESC
        MOV     WORD PTR TS1LDT,AX
        MOV     AX,OFFSET SRCDESC
        MOV     WORD PTR TS1DS,AX
        MOV     AX,OFFSET DSTDESC
        MOV     WORD PTR TS1ES,AX
        MOV     AX,OFFSET TS1CODE
        MOV     WORD PTR TS1CS,AX
        MOV     AX,0068H
        MOV     TS1IOM,AX
        POP     ES
        POP     DS
        RET
        ASSUME  DS:DATA
;
;       TASK 2関係のDESCRIPTORとTSSの初期化
;
SETUP2:
        PUSH    DS
        PUSH .  ES
        MOV     AX,SEG GDT
        MOV     ES,AX
;
        MOV     AX,0FFFFH                       ; TASK 2 CODE
        MOV     BL,0                ; DPL
        MOV     BH,1AH              ; CODE READ可能
        MOV     CX,LDLO
        MOV     DL,0                ; LIMIT上位
        MOV     DH,40H              ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,LDHI
        MOV     DI,OFFSET TS2CODE
        CALL    SETDSC
;
        MOV     AX,0FFFFH                       ; TASK 2 DATA
        MOV     BL,0                ; DPL=0
        MOV     BH,12H              ; DATA R/W
        MOV     CX,LDLO
        MOV     DL,0FH              ; 1MB
        MOV     DH,40H              ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,LDHI
        MOV     DI,OFFSET DSTDESC
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1                     ; 特権LEVEL 0用STACK
        MOV     BL,0                ; DPL
        MOV     BH,12H              ; DATA R/W
        MOV     CX,STK20            ; BASE下位
        MOV     DL,0                ; LIMIT上位
        MOV     DH,40H              ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0                ; BASE上位
        MOV     DI,OFFSET TS2STK0
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1                     ; 特権LEVEL 1用STACK
        MOV     BL,1                ; DPL
        MOV     BH,12H              ; DATA R/W
        MOV     CX,STK21            ; BASE下位
        MOV     DL,0                ; LIMIT上位
        MOV     DH,40H              ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0                ; BASE上位
        MOV     DI,OFFSET TS2STK1
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1                     ; 特権LEVEL 2用のSTACK
        MOV     BL,2                ; DPL
        MOV     BH,12H              ; DATA R/W
        MOV     CX,STK22            ; BASE下位
        MOV     DL,0                ; LIMIT上位
        MOV     DH,40H              ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0                ; BASE上位
        MOV     DI,OFFSET TS2STK2
        CALL    SETDSC
;
        MOV     AX,STKSIZ-1                     ; 初期使用のSTACK
        MOV     BL,0                ; DPL
        MOV     BH,12H              ; DATA R/W
        MOV     CX,STK2TOP          ; BASE下位
        MOV     DL,0                ; LIMIT上位
        MOV     DH,40H              ; GRANULE=BYTE,DEFAULT=32
        MOV     SI,0                ; BASE上位
        MOV     DI,OFFSET ST2DESC
        CALL    SETDSC
;
        MOV     AX,OFFSET TS2END-1        ; TSS
        MOV     BL,0                ; DPL
        MOV     BH,09H              ; 使用可能な386TSS
        MOV     CX,SEG TS2TSS       ; BASE下位
        MOV     DL,0                ; LIMIT上位
        MOV     DH,0                ; GRANULE=BYTE
        MOV     SI,0                ; BASE上位
        MOV     DI,OFFSET TS2DESC
        CALL    SETDSC
;
        MOV     AX,SEG TS2TSS                   ; TSS SEGMENTの初期設定
        MOV     ES,AX
        XOR     AX,AX
        MOV     CX,OFFSET TS2END
        SHR     CX,1
        MOV     DI,0
        CLD
        REP     STOSW
        MOV     AX,ES
        MOV     DS,AX
        ASSUME  DS:TS2TSS
        MOV     AX,STKSIZ                       ; SPの初期値
        MOV     WORD PTR TS2ESP0,AX
        MOV     WORD PTR TS2ESP1,AX
        MOV     WORD PTR TS2ESP2,AX
        MOV     WORD PTR TS2ESP,AX
        MOV     AX,0202H                        ; 割込許可、IOPL=0
        MOV     WORD PTR TS2EF,AX
        MOV     AX,OFFSET TS2STK0               ; 各SELECTORの初期値
        MOV     WORD PTR TS2SS0,AX
        MOV     AX,OFFSET TS2STK1
        MOV     WORD PTR TS2SS1,AX
        MOV     AX,OFFSET TS2STK2
        MOV     WORD PTR TS2SS2,AX
        MOV     AX,OFFSET ST2DESC
        MOV     WORD PTR TS2SS,AX
        MOV     AX,OFFSET LDTDESC
        MOV     WORD PTR TS2LDT,AX
        MOV     AX,OFFSET DSTDESC
        MOV     WORD PTR TS2DS,AX
        MOV     WORD PTR TS2ES,AX
        MOV     AX,OFFSET COMDESC
        MOV     WORD PTR TS2FS,AX
        MOV     AX,OFFSET BUFDESC
        MOV     WORD PTR TS2GS,AX
```

メモリ管理用語集

デアロケート（deallocate）　アプリケーションに与えた論理ページを解放し，ほかのアプリケーションからも使えるようにすること．

命令を実行するだけでよい．ただし，EFLAGSのNTビットを変更しているならば，NTビットを1に戻してからIRET命令を実行しなければならない．終了時のレジスタの内容は，コマンドインタプリタのRコマンドで表示することが可能である．

リスト2は，タスク2として動作可能なファイルダンプのプログラムである．このプログラムをP386の実行ファイルに変換するには，リスト3のバッチプログラムを使用する．

実際に実行するには，

```
A>P386 TF386 XXX
```

(XXXはダンプするファイル名)

とMS-DOSのコマンドラインから入力するか，

```
A>P386
>TF386  XXX
```

というように，P386のコマンドラインから入力する．このとき増設メモリの内容を破壊するので，増設メモリを使用するソフトウェアやデバイスドライバとの同居は不可である．

このサンプルプログラムは，MS-DOSのプログラムとほとんど変わらないが，ファイル名をBUFセグメントに配置したり，ディスクからリードされるデータがBUFセグメントに固定的に置かれるなど多少不自由である．P386自体を極力簡単なものにしたため，いたしかたない．元気のある読者は，ぜひ改造を試みてほしい．

(本文では80386上で動作するP386だけを取り上げたが，付録ディスク内には80286上で動作するP286，TF286も収録してあるので，80286マシンをおもちの方はこちらを試していただきたい．)

```
        MOV     AX,OFFSET TS2CODE
        MOV     WORD PTR TS2CS,AX
        MOV     AX,0068H                        ; IO MAPのOFFSET
        MOV     TS2IOM,AX
        POP     ES
        POP     DS
        RET
        ASSUME  DS:DATA
;
;       COM SEGMENTのDUMP
;
COMDSP:
        PUSH    DS
        PUSH    ES
        MOV     AX,SEG DATA
        MOV     DS,AX
        MOV     AX,SEG COM
        MOV     ES,AX
        MOV     SI,OFFSET MSGR0
        MOV     DI,OFFSET CR0SAV
        MOV     CX,13
COMDSP3:
        MOV     DX,3
COMDSP2:
        PUSH    SI
        MOV     SI,OFFSET MSG0
        CALL    MSGOUT
        POP     SI
        CALL    MSGOUT
        MOV     AL,ES:3[DI]
        CALL    HBOUT
        MOV     AL,ES:2[DI]
        CALL    HBOUT
        MOV     AL,ES:1[DI]
        CALL    HBOUT
        MOV     AL,ES:[DI]
        CALL    HBOUT
        ADD     DI,4
        DEC     CX
        JZ      COMDSP1
        DEC     DX
        JNZ     COMDSP2
        PUSH    SI
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        POP     SI
        JMP     SHORT COMDSP3
COMDSP1:
        PUSH    SI
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        POP     SI
        MOV     CX,8
COMDSP6:
        MOV     DX,4
COMDSP5:
        PUSH    SI
        MOV     SI,OFFSET MSG0
        CALL    MSGOUT
        POP     SI
        CALL    MSGOUT
        MOV     AL,ES:1[DI]
        CALL    HBOUT
        MOV     AL,ES:[DI]
        CALL    HBOUT
        ADD     DI,4
        DEC     CX
        JZ      COMDSP4
        DEC     DX
        JNZ     COMDSP5
        PUSH    SI
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        POP     SI
        JMP     SHORT COMDSP6
COMDSP4:
        PUSH    SI
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        POP     SI
        MOV     CX,2
COMDSP7:
        PUSH    SI
        MOV     SI,OFFSET MSG0
        CALL    MSGOUT
        POP     SI
        CALL    MSGOUT
        MOV     AL,ES:5[DI]
        CALL    HBOUT
        MOV     AL,ES:4[DI]
        CALL    HBOUT
        MOV     AL,ES:3[DI]
        CALL    HBOUT
        MOV     AL,ES:2[DI]
        CALL    HBOUT
        MOV     AL,ES:1[DI]
        CALL    HBOUT
        MOV     AL,ES:[DI]
        CALL    HBOUT
        ADD     DI,8
        LOOP    COMDSP7
        MOV     DI,OFFSET ERRSAV
        MOV     AX,ES:[DI]
        AND     AX,AX
        JZ      COMDSP8
        ADD     DI,4
        PUSH    SI
        MOV     SI,OFFSET MSG0
        CALL    MSGOUT
        MOV     SI,OFFSET MSGR23
        CALL    MSGOUT
        POP     SI
        MOV     AL,ES:1[DI]
        CALL    HBOUT
        MOV     AL,ES:[DI]
        CALL    HBOUT
COMDSP8:
        MOV     SI,OFFSET CRLF
        CALL    MSGOUT
        POP     ES
        POP     DS
        RET
;
;       REAL MODEのSEGMENT SELECTORの値をLINEAR ADDRESSに変換
;       DX:AX=(DX:AX) * 16
;
LINEAR:
        ADD     AX,AX
        ADC     DX,DX
        ADD     AX,AX
        ADC     DX,DX
        ADD     AX,AX
        ADC     DX,DX
        ADD     AX,AX
        ADC     DX,DX
        RET
```


メモリ管理用語集

TSR（Terminate and Stay Resident）　主記憶常駐終了型プログラムを指す．タイマやキー割り込みで起動する．

```
;
;       SEGMENT DESCRIPTORの初期化
;       DL:AX=LIMIT, BL=DPL, DH:BH=TYPE, SI:CX=BASE/16
;       ES:DI=DESCRIPTOR POSITION
;
SETDSC:                                 ; 386 VERSION
        CLD
        STOSW                           ; LIMIT[0-15]
        MOV     AX,CX
        MOV     CX,4
SETDSC1:
        SHL     AX,1
        RCL     SI,1
        LOOP    SETDSC1
        STOSW                           ; BASE[0-15]
        MOV     AX,SI
        STOSB                           ; BASE[16-23]
        MOV     AL,BL
        MOV     CL,5
        SHL     AL,CL
        OR      AL,BH
        OR      AL,80H
        STOSB                           ; DPL AND TYPE
        XOR     AX,AX
        MOV     AL,DL                   ; LIMIT[16-19]
        AND     AL,0FH
        AND     DH,0C0H                 ; GRANULE AND DEFAULT SIZE
        OR      AL,DH
        STOSB
        MOV     AX,SI
        MOV     AL,AH
        STOSB
        RET
;
;       GATE DESCRIPTORの初期化
;       DX:AX=OFFSET, BL=DPL, BH=TYPE, CX=SELECTOR
;       SI=WORD COUNT, ES:DI=GATE POSITION
;
SETGAT:                                 ; 386 VERSION
        CLD
        STOSW                           ; OFFSET
        MOV     AX,CX
        STOSW                           ; SELECTOR
        MOV     AX,SI
        AND     AL,1FH
        STOSB                           ; WORD COUNT
        MOV     AL,BL
        MOV     CL,5
        SHL     AL,CL
        OR      AL,BH
        OR      AL,80H
        STOSB                           ; DPLとTYPE
        MOV     AX,DX
        STOSW
        RET
;
;       CR CODEまでの1行分の文字列をSTDINより入力
;
MSGIN:
        MOV     BYTE PTR CBUFLEN,254
        MOV     DX,OFFSET CBUFLEN
        MOV     AH,0AH
        INT     MSDOS
        MOV     SI,OFFSET CBUF
        MOV     AL,CINPLEN
        MOV     AH,0
        ADD     SI,AX
        MOV     [SI],AH
        RET
;
;       NUL CODEまでの文字列をSTDOUTへ出力
;
MSGOUT:
        MOV     AL,[SI]
        INC     SI
        AND     AL,AL
        JZ      MSGOUT1
        CALL    CO
        JMP     SHORT MSGOUT
MSGOUT1:
        RET
;
;       1文字をSTDOUTへ出力
;
CO:
        CMP     AL,0FFH
        JZ      CO1
        PUSH    AX
        PUSH    DX
        MOV     DL,AL
        MOV     AH,06H
        INT     MSDOS
        POP     DX
        POP     AX
CO1:
        RET
;
;       ALの下位NIBBLEを16進数の文字へ変換してALに格納
;
HJN:
        AND     AL,0FH
        CMP     AL,09H
        JBE     HJN1
        ADD     AL,07H
HJN1:
        ADD     AL,30H
        RET
;
;       ALの1BYTEを16進数2文字でSTDOUTへ出力
;
HBOUT:
        PUSH    AX
        PUSH    CX
        PUSH    AX
        MOV     CL,4
        SHR     AL,CL
        CALL    HJN
        CALL    CO
        POP     AX
        CALL    HJN
        CALL    CO
        POP     CX
        POP     AX
        RET
;
;       PATHの形式上許される文字以外の文字が現れるまで文字を転送
;       DS:SI ---> DS:DI (FILE拡張子を除く)
;
FNSET:
        MOV     CL,60
        MOV     AL,1[SI]
        CMP     AL,':'
        JNZ     FNSET1
        MOV     AL,[SI]
        MOV     [DI],AL
        INC     SI
        INC     DI
        MOV     AL,[SI]
        MOV     [DI],AL
        INC     SI
        INC     DI
        SUB     CL,2
FNSET1:
        MOV     AL,[SI]
        CMP     AL,'.'
        JZ      FNSET3
        CALL    NEAR PTR DLMT
        JZ      FNSET3
        MOV     [DI],AL
        INC     SI
        INC     DI
        DEC     CL
        JNZ     FNSET1
FNSET3:
        XOR     AL,AL
        MOV     [DI],AL
        RET
;
;       ファイル拡張子の形式上許される以外の文字が現れるまで文字を転送
;       DS:SI ---> DS:DI
;
FTSET:
        MOV     AL,[SI]
        CMP     AL,'.'
        JZ      FTSET1
FTSET2:
        XOR     AL,AL
        MOV     [DI],AL
        RET
FTSET1:
        MOV     [DI],AL
        INC     SI
        INC     DI
        MOV     CL,3
FTSET3:
        MOV     AL,[SI]
        CALL    NEAR PTR DLMT
        JZ      FTSET2
        MOV     [DI],AL
        INC     SI
        INC     DI
        DEC     CL
        JNZ     FTSET3
        JMP     SHORT FTSET2
;
;       ALの文字がPATHの中で許される文字かどうかを検査する
;       FLAGS=NZ:許される文字, Z:許されない文字
;
DLMT:
        CMP     AL,' '
        JNC     DLMT2
        PUSH    AX
        XOR     AX,AX
        POP     AX
        RET
DLMT2:
        CMP     AL,' '
        JZ      DLMT1
        CMP     AL,'?'
        JZ      DLMT1
        CMP     AL,'*'
        JZ      DLMT1
        CMP     AL,'/'
        JZ      DLMT1
        CMP     AL,','
        JZ      DLMT1
        CMP     AL,'.'
        JZ      DLMT1
        CMP     AL,':'
        JZ      DLMT1
        CMP     AL,';'
        JZ      DLMT1
        CMP     AL,'['
        JZ      DLMT1
        CMP     AL,']'
        JZ      DLMT1
        CMP     AL,'<'
        JZ      DLMT1
        CMP     AL,'>'
        JZ      DLMT1
        CMP     AL,'"'
        JZ      DLMT1
        CMP     AL,'='
        JZ      DLMT1
```

メモリ管理用語集

DPMI　1MBを超えるメモリ空間を複数のアプリケーションから利用するためにMicrosoftをはじめとする米国大手ソフトベンダ11社で提唱された規格.

```
        CMP     AL,'+'
        JZ      DLMT1
        CMP     AL,'|'
DLMT1:
        RET
;
;       DS:SIの文字列の先頭にDS:DIの文字列が存在するかを調べる
;       FLAGS=NZ:存在しない, Z:存在する
;
KYWD:
        MOV     AL,[SI]
        MOV     AH,[DI]
        AND     AH,AH
        JZ      KYWD1
        CMP     AL,'a'
        JC      KYWD2
        CMP     AL,'z'
        JA      KYWD2
        SUB     AL,20H
KYWD2:
        CMP     AL,AH
        JNZ     KYWD1
        INC     SI
        INC     DI
        JMP     SHORT KYWD
KYWD1:
        RET
;
;       DS:SIの文字列でSPACEとTAB以外の最初の文字を探す
;
SKPBLN:
        MOV     AL,[SI]
        INC     SI
        CMP     AL,' '
        JZ      SKPBLN
        CMP     AL,09H
        JZ      SKPBLN
        DEC     SI
        RET
TSK0END EQU     $
;
TEXT    ENDS
;
; プロテクトモードでの割込処理 (特権LEVEL 0, TASK 1,2共用)
;
INTP    SEGMENT
        ASSUME  CS:INTP,DS:COM
INT00:
        IENTRY
        PUSH    EAX
        MOV     AL,0FFH
INTHND:
        MOV     AH,0
        CWDE
        PUSH    EAX
        PUSH    DS
        MOV     EAX,OFFSET COMDESC      ; DS=COM SEGMENT
        MOV     DS,EAX
        POP     EAX                     ; 割込時のDS,ES,FS,GSをSAVE
        MOV     DSSAV,EAX
        MOV     ESSAV,ES
        MOV     FSSAV,FS
        MOV     GSSAV,GS
        POP     EAX                     ; 割込VECTOR番号をSAVE
        MOV     FUNCNO,EAX
        MOV     EBXSAV,EBX              ; 割込時のBX,CX,DX,SI,BPをSAVE
        MOV     ECXSAV,ECX
        MOV     EDXSAV,EDX
        MOV     ESISAV,ESI
        MOV     EDISAV,EDI
        MOV     EBPSAV,EBP
        POP     EBX                     ; 割込時のAXをSAVE
        MOV     EAXSAV,EBX
        XOR     EBX,EBX                 ; ERRSAV,HINTNOを初期化
        MOV     ERRSAV,EBX
        MOV     HINTNO,EBX
        MOV     EBP,ESP
        CMP     AL,08H
        JC      SHORT INTHND1
        CMP     AL,18H
        JNC     SHORT INTHND1
        PUSH    EAX                     ; HARDWARE割込の有無を調査
        MOV     AL,0BH                  ; PIC MASTERのISRをリード
        OUT     00H,AL
        JMP     SHORT $+2
        JMP     SHORT $+2
        IN      AL,00H
        AND     AL,AL
        JZ      SHORT INTHND3           ; HARDWAREの割込なし
        MOV     BL,8                    ; ISRの値をVECTOR番号に変換
        MOV     BH,0
        CWDE
INTHND5:
        SHR     AL,1
        JC      SHORT INTHND4
        INC     EBX
        JMP     SHORT INTHND5
INTHND4:
        CMP     BL,15
        JNZ     SHORT INTHND6           ; PIC MASTERからの割込
        MOV     AL,0BH                  ; PIC SLAVEのISRをリード
        OUT     08H,AL
        JMP     SHORT $+2
        JMP     SHORT $+2
        IN      AL,08H
        AND     AL,AL
        JZ      SHORT INTHND3           ; PIC SLAVEからの割込なし
        MOV     BL,16                   ; ISRの値をVECTOR番号に変換
        MOV     BH,0
        CWDE
INTHND7:
        SHR     AL,1
        JC      SHORT INTHND6
        INC     EBX
        JMP     SHORT INTHND7
INTHND6:
        OR      BH,80H                  ; HADWARE割込のVECTOR番号をSAVE
        MOV     HINTNO,EBX
INTHND3:
        POP     EAX
;
INTHND1:
        CMP     AL,8                    ; ERROR CODEの有無を調査
        JZ      INTHND8
        CMP     AL,10
        JC      INTHND9                 ; ERROR CODEなし
        CMP     AL,15
        JNC     INTHND9                 ; ERROR CODEなし
INTHND8:
        MOV     EBX,HINTNO
        AND     EBX,EBX
        JZ      SHORT INTHND10          ; 割込なし、ERROR CODEあり
        CMP     AL,BL
        JZ      SHORT INTHND9           ; 例外より割込を優先
INTHND10:
        XOR     EAX,EAX                 ; ERROR CODEをPOPしてSAVE
        DEC     EAX
        MOV     ERRSAV,EAX
        MOV     EAX,0[EBP]
        MOV     ERRCOD,EAX
        ADD     ESP,4
        MOV     EBP,ESP
INTHND9:
        MOV     EAX,0[EBP]              ; 割込時のIP,CS,FLAGS,SP,SS
        MOV     EIPSAV,EAX              ; をSAVE
        MOV     EAX,4[EBP]
        MOV     CSSAV,EAX
        MOV     EAX,8[EBP]
        MOV     EFSAV,EAX
        MOV     ESPSAV,ESP
        MOV     SSSAV,SS
        SGDT    FWORD PTR GDTSAV        ; 割込時のGDTR,IDTRをSAVE
        SIDT    FWORD PTR IDTSAV
        SLDT    WORD PTR LDTSAV         ; 割込時のLDT,TRのSELECTORをSAVE
        STR     SI
        AND     ESI,0FFFFH
        MOV     TRSAV,ESI
        MOV     EAX,CR0                 ; 割込時のCR0,CR2,CR3をSAVE
        MOV     CR0SAV,EAX
        MOV     EAX,CR3
        MOV     CR3SAV,EAX
        MOV     EAX,CR2
        MOV     CR2SAV,EAX
        MOV     EAX,OFFSET GDTDESC      ; TSS DESCRIPTORのBUSY BITをRESET
        MOV     DS,EAX
        MOV     AL,5[ESI]
        AND     AL,0FDH
        MOV     5[ESI],AL
        CPURST                          ; CPUのRESET
INTEND  EQU     $
INTP    ENDS
;
;       TASK 1 (特権LEVEL 0, MEMORY DATAの転送)
;
TASK1   SEGMENT
        ASSUME  CS:TASK1
MOVE:
        MOV     EDX,ECX
        SHR     ECX,2
MOVE1:
        REP     MOVSD
        MOV     ECX,EDX
        AND     ECX,03H
        JZ      SHORT MOVE2
        REP     MOVSB
MOVE2:
        IRETD
        JMP     SHORT MOVE
TSK1END EQU     $
TASK1   ENDS
;
;       REAL MODE用のデータ
;
DATA    SEGMENT
MSG0    DB      '  ',0
MSG1    DB      '** MEMORYを確保できません。
CRLF    DB      CR,LF,0
MSG2    DB      '>',0
MSG3    DB      '** FILE名を指定してください。
        DB      CR,LF,0
MSG4    DB      '** ',0
MSG5    DB      'はOPENできません。
        DB      CR,LF,0
MSG6    DB      'にREAD ERRORが発生しました。
        DB      CR,LF,0
CMD0    DB      'Q',0
CMD1    DB      'R',0
FTDFLT  DB      '.BIN',0
MSGT0   DB      '** DIVISION **',0
MSGT1   DB      '** DEBUG **',0
MSGT2   DB      '** NMI **',0
MSGT3   DB      '** BREAK POINT **',0
MSGT4   DB      '** OVERFLOW **',0
MSGT5   DB      '** BOUNDARY **',0
MSGT6   DB      '** OP CODE **',0
MSGT7   DB      '** NDP UNAVAILABLE **',0
MSGT8   DB      '** DOUBLE FAULT **',0
MSGT9   DB      '** NDP OVERRUN **',0
MSGT10  DB      '** INVALID TSS **',0
MSGT11  DB      '** SEGMENT ABSENT **',0
MSGT12  DB      '** INVALID SS **',0
MSGT13  DB      '** GENERAL PROTECTION **',0
MSGT14  DB      '** PAGING **',0
MSGT16  DB      '** NDP OPERATION **',0
```


メモリ管理用語集
ディスクリプタ　プロテクトモードにおいてアドレス参照はすべて間接的に行われる．CPUはディスクリプタテーブルの値を参照してアドレスを決定する．

```
MSGR0   DB      'CR0   = ',0
MSGR1   DB      'CR2   = ',0
MSGR2   DB      'CR3   = ',0
MSGR3   DB      'EIP   = ',0
MSGR4   DB      'EFLAGS= ',0
MSGR5   DB      'EAX   = ',0
MSGR6   DB      'EBX   = ',0
MSGR7   DB      'ECX   = ',0
MSGR8   DB      'EDX   = ',0
MSGR9   DB      'ESI   = ',0
MSGR10  DB      'EDI   = ',0
MSGR11  DB      'EBP   = ',0
MSGR12  DB      'ESP   = ',0
MSGR13  DB      'CS    = ',0
MSGR14  DB      'DS    = ',0
MSGR15  DB      'ES    = ',0
MSGR16  DB      'FS    = ',0
MSGR17  DB      'GS    = ',0
MSGR18  DB      'SS    = ',0
MSGR19  DB      'TR    = ',0
MSGR20  DB      'LDT   = ',0
MSGR21  DB      'GDT   = ',0
MSGR22  DB      'IDT   = ',0
MSGR23  DB      'ERROR = ',0
        EVEN
BUFTOP  DW      1 DUP(?)        ; BUF SEGMENTのSELECTOR
LDTTOP  DW      1 DUP(?)        ; LDT SEGMENTのSELECTOR
STK00   DW      1 DUP(?)        ; 以下各STACK SEGMENTのSELECTOR
STK01   DW      1 DUP(?)        ; "
STK02   DW      1 DUP(?)        ; "
STK10   DW      1 DUP(?)        ; "
STK11   DW      1 DUP(?)        ; "
STK12   DW      1 DUP(?)        ; "
STK20   DW      1 DUP(?)        ; "
STK21   DW      1 DUP(?)        ; "
STK22   DW      1 DUP(?)        ; "
STK0TOP DW      1 DUP(?)        ; "
STK1TOP DW      1 DUP(?)        ; "
STK2TOP DW      1 DUP(?)        ; "
GDTORG  DQ      1 DUP(?)        ; 本PROGRAM実行開始時点でのGDTR
IDTORG  DQ      1 DUP(?)        ; 本PROGRAM実行開始時点でのIDTR
GDTR    DQ      1 DUP(?)        ; PROTECTED MODE用GDT DESCRIPTOR
IDTR    DQ      1 DUP(?)        ; PROTECTED MODE用IDT DESCRIPTOR
LDTRAP  DW      1 DUP(?)        ; PROGRAMのLOAD中ONになるFLAG
STAMOD  DW      1 DUP(?)        ; 0であればCOMMAND FILE実行後直ちに終了
CMDHND  DW      1 DUP(?)        ; COMMAND FILEのFILE HANDLE
CMDEOF  DW      1 DUP(?)        ; COMMAND FILEがEOFに到達したことのFLAC
CMDBYT  DD      1 DUP(?)        ; BUF SEGMENTから転送すべきBYTE数
CMDFN   DB      66 DUP(?)       ; COMMAND FILEのFILE名
CBUFLEN DB      1 DUP(?)        ; CBUFの長さ
CINPLEN DB      1 DUP(?)        ; CBUFに入力された文字数(DOSが格納)
CBUF    DB      256 DUP(?)      ; COMMAND LINE入力用
DATA    ENDS
;
;       GLOBAL DESCRIPTOR TABLE
;
GDT     SEGMENT
NULL    DQ      1 DUP(?)        ; NULL SELECTOR
GDTDESC DQ      1 DUP(?)        ; GDT SEGMENT ALIAS (DATA)
IDTDESC DQ      1 DUP(?)        ; IDT SEGMENT ALIAS (DATA)
INTDESC DQ      1 DUP(?)        ; INTP SEGMENT (CODE)
LDTDESC DQ      1 DUP(?)        ; LDT SYSTEM SEGMENT (LDT)
COMDESC DQ      1 DUP(?)        ; COM SEGMENT (DATA)
BUFDESC DQ      1 DUP(?)        ; BUF SEGMENT (DATA)
TS0CODE DQ      1 DUP(?)        ; TASK 0 CODE (CODE)
TS1CODE DQ      1 DUP(?)        ; TASK 1 CODE (CODE)
TS2CODE DQ      1 DUP(?)        ; TASK 2 CODE (CODE)
TS0STK0 DQ      1 DUP(?)        ; TASK 0 STACK LEVEL 0 (DATA)
TS1STK0 DQ      1 DUP(?)        ; TASK 1 STACK LEVEL 0 (DATA)
TS2STK0 DQ      1 DUP(?)        ; TASK 2 STACK LEVEL 0 (DATA)
TS0STK1 DQ      1 DUP(?)        ; TASK 0 STACK LEVEL 1 (DATA)
TS1STK1 DQ      1 DUP(?)        ; TASK 1 STACK LEVEL 1 (DATA)
TS2STK1 DQ      1 DUP(?)        ; TASK 2 STACK LEVEL 1 (DATA)
TS0STK2 DQ      1 DUP(?)        ; TASK 0 STACK LEVEL 2 (DATA)
TS1STK2 DQ      1 DUP(?)        ; TASK 1 STACK LEVEL 2 (DATA)
TS2STK2 DQ      1 DUP(?)        ; TASK 2 STACK LEVEL 2 (DATA)
ST0DESC DQ      1 DUP(?)        ; TASK 0 DEFAULT STACK (DATA)
ST1DESC DQ      1 DUP(?)        ; TASK 1 DEFAULT STACK (DATA)
ST2DESC DQ      1 DUP(?)        ; TASK 2 DEFAULT STACK (DATA)
TS0DESC DQ      1 DUP(?)        ; TASK 0 TSS SYSTEM SEGMENT (TSS)
TS1DESC DQ      1 DUP(?)        ; TASK 1 TSS SYSTEM SEGMENT (TSS)
TS2DESC DQ      1 DUP(?)        ; TASK 2 TSS SYSTEM SEGMENT (TSS)
SRCDESC DQ      1 DUP(?)        ; TEMPORARY USE (DATA)
DSTDESC DQ      1 DUP(?)        ; TEMPORARY USE (DATA)
GDTEND  EQU     $
GDT     ENDS
;
;       INTERRUPT DESCRIPTOR TABLE
;
IDT     SEGMENT
IDTBGN  DQ      256 DUP(?)
IDTEND  EQU     $
IDT     ENDS
;
;       TASK 0用TSS
;
TS0TSS  SEGMENT
TS0LNK  DD      1 DUP(?)
TS0ESP0 DD      1 DUP(?)
TS0SS0  DD      1 DUP(?)
TS0ESP1 DD      1 DUP(?)
TS0SS1  DD      1 DUP(?)
TS0ESP2 DD      1 DUP(?)
TS0SS2  DD      1 DUP(?)
TS0CR3  DD      1 DUP(?)
TS0EIP  DD      1 DUP(?)
TS0EF   DD      1 DUP(?)
TS0EAX  DD      1 DUP(?)
TS0ECX  DD      1 DUP(?)
TS0EDX  DD      1 DUP(?)
TS0EBX  DD      1 DUP(?)
TS0ESP  DD      1 DUP(?)
TS0EBP  DD      1 DUP(?)
TS0ESI  DD      1 DUP(?)
TS0EDI  DD      1 DUP(?)
TS0ES   DD      1 DUP(?)
TS0CS   DD      1 DUP(?)
TS0SS   DD      1 DUP(?)
TS0DS   DD      1 DUP(?)
TS0FS   DD      1 DUP(?)
TS0GS   DD      1 DUP(?)
TS0LDT  DD      1 DUP(?)
TS0TRC  DW      1 DUP(?)
TS0IOM  DW      1 DUP(?)
        DB      32 DUP(?)
TS0END  EQU     $
TS0TSS  ENDS
;
;       TASK 1用TSS
;
TS1TSS  SEGMENT
TS1LNK  DD      1 DUP(?)
TS1ESP0 DD      1 DUP(?)
TS1SS0  DD      1 DUP(?)
TS1ESP1 DD      1 DUP(?)
TS1SS1  DD      1 DUP(?)
TS1ESP2 DD      1 DUP(?)
TS1SS2  DD      1 DUP(?)
TS1CR3  DD      1 DUP(?)
TS1EIP  DD      1 DUP(?)
TS1EF   DD      1 DUP(?)
TS1EAX  DD      1 DUP(?)
TS1ECX  DD      1 DUP(?)
TS1EDX  DD      1 DUP(?)
TS1EBX  DD      1 DUP(?)
TS1ESP  DD      1 DUP(?)
TS1EBP  DD      1 DUP(?)
TS1ESI  DD      1 DUP(?)
TS1EDI  DD      1 DUP(?)
TS1ES   DD      1 DUP(?)
TS1CS   DD      1 DUP(?)
TS1SS   DD      1 DUP(?)
TS1DS   DD      1 DUP(?)
TS1FS   DD      1 DUP(?)
TS1GS   DD      1 DUP(?)
TS1LDT  DD      1 DUP(?)
TS1TRC  DW      1 DUP(?)
TS1IOM  DW      1 DUP(?)
        DB      32 DUP(?)
TS1END  EQU     $
TS1TSS  ENDS
;
;       TASK 2用TSS
;
TS2TSS  SEGMENT
TS2LNK  DD      1 DUP(?)
TS2ESP0 DD      1 DUP(?)
TS2SS0  DD      1 DUP(?)
TS2ESP1 DD      1 DUP(?)
TS2SS1  DD      1 DUP(?)
TS2ESP2 DD      1 DUP(?)
TS2SS2  DD      1 DUP(?)
TS2CR3  DD      1 DUP(?)
TS2EIP  DD      1 DUP(?)
TS2EF   DD      1 DUP(?)
TS2EAX  DD      1 DUP(?)
TS2ECX  DD      1 DUP(?)
TS2EDX  DD      1 DUP(?)
TS2EBX  DD      1 DUP(?)
TS2ESP  DD      1 DUP(?)
TS2EBP  DD      1 DUP(?)
TS2ESI  DD      1 DUP(?)
TS2EDI  DD      1 DUP(?)
TS2ES   DD      1 DUP(?)
TS2CS   DD      1 DUP(?)
TS2SS   DD      1 DUP(?)
TS2DS   DD      1 DUP(?)
TS2FS   DD      1 DUP(?)
TS2GS   DD      1 DUP(?)
TS2LDT  DD      1 DUP(?)
TS2TRC  DW      1 DUP(?)
TS2IOM  DW      1 DUP(?)
        DB      32 DUP(?)
TS2END  EQU     $
TS2TSS  ENDS
;
;       REAL MODE用STACK SEGMENT
;
STACK   SEGMENT
        DW      512 DUP(?)
ENDSEG  EQU     $
STACK   ENDS
;
        END     START

```

リスト2：TF386.ASM

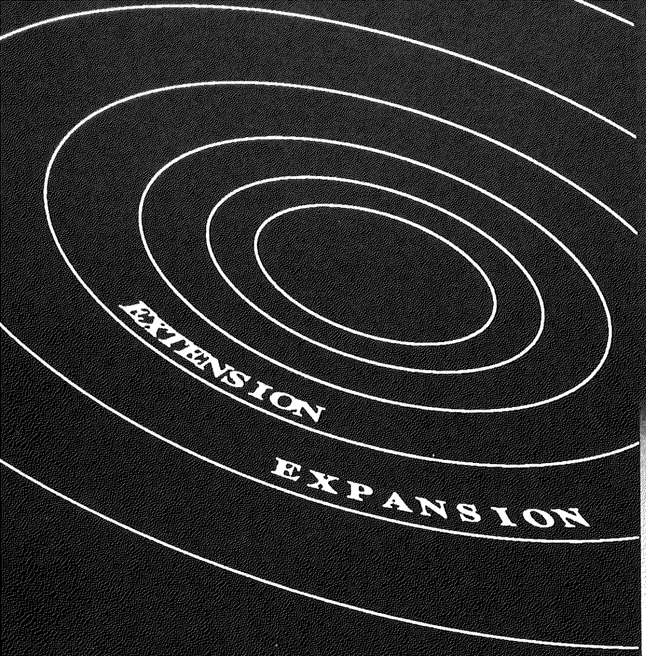
```
;
;       FILE DUMP (PROTECTED MODEで実行)
;
        .386
```

```
;
TEXT    SEGMENT PARA USE32 'CODE'
TEXT    ENDS
;
CR      EQU     0DH
LF      EQU     0AH
;
MSDOS   EQU     21H
COUT    EQU     02H
OPN     EQU     3DH
CLS     EQU     3EH
READ    EQU     3FH
;
DBUF    EQU     0000H           ; (FILE DATA読込位置) BUF SEGMENT
FNAM    EQU     8000H           ; (FILE NAME格納位置) BUF SEGMENT
;
TEXT    SEGMENT
        ASSUME  CS:TEXT,DS:TEXT
;
DUMP:
        PUSH    DS              ; FILE名が指定されているか確認
        MOV     EAX,FS
        MOV     DS,EAX
        XOR     ESI,ESI
        CALL    SKPBLN
        AND     AL,AL
        POP     DS
        JZ      ERR1            ; FILE名が指定されていない
        PUSH    DS
        PUSH    ES
        MOV     EAX,FS          ; FILE名をCOM SEGMENTからBUF SEGMENT
        MOV     DS,EAX          ; へ転送
        MOV     EAX,GS
        MOV     ES,EAX
        MOV     EDI,FNAM
        CLD
DUMP1:
        LODSB
        STOSB
        AND     AL,AL
        JNZ     DUMP1
        POP     ES
        POP     DS
        XOR     EAX,EAX
        MOV     ADRS,EAX        ; ADRS = 0
        MOV     EOF,EAX         ; EOF  = 0
        MOV     AH,OPN          ; 指定されたFILEをOPEN
        MOV     AL,0
        MOV     EDX,FNAM
        INT     MSDOS
        JC      ERR2            ; OPENできない
        MOV     FHAND,EAX
DUMP6:
        MOV     EBX,FHAND
        MOV     ECX,8000H       ; 32KBの読込を要求
        XOR     EDX,EDX         ; BUF SEGMENTの先頭へ
        MOV     AH,READ
        INT     MSDOS
        JC      ERR3            ; 読込ERROR発生
        AND     EAX,0FFFFH
        MOV     RLEN,EAX        ; RLEN=実際に読込まれたBYTE数
        CMP     EAX,8000H       ; まだ読込める可能性があるか
        JZ      SHORT DUMP2     ; YES
        XOR     EAX,EAX         ; EOFに達した
        DEC     EAX
        MOV     EOF,EAX
DUMP2:
        XOR     ESI,ESI
DUMP5:
        MOV     AL,BYTE PTR ADRS+3      ; 各行の先頭にFILE内OFFSETを表示
        CALL    HBOUT
        MOV     AL,BYTE PTR ADRS+2
        CALL    HBOUT
        MOV     AL,BYTE PTR ADRS+1
        CALL    HBOUT
        MOV     AL,BYTE PTR ADRS
        CALL    HBOUT
        MOV     AL,' '
        CALL    CO
        MOV     BL,16           ; 1行に16 BYTES分表示
DUMP4:
        MOV     AL,' '
        CALL    CO
        MOV     AL,GS:[ESI]     ; BUF SEGMENTに読込まれたDATAを処理
        INC     ESI
        CALL    HBOUT
        INC     DWORD PTR ADRS  ; ADRS = ADRS + 1
        DEC     DWORD PTR RLEN  ; RLEN = RLEN - 1
        JZ      DUMP3           ; 読込まれたデータを全部処理した
        DEC     BL
        JNZ     DUMP4
        MOV     AL,CR           ; 1行の終わりに到達、改行
        CALL    CO
        MOV     AL,LF
        CALL    CO
        JMP     SHORT DUMP5
DUMP3:
        MOV     AL,CR
        CALL    CO
        MOV     AL,LF
        CALL    CO
        MOV     EAX,EOF         ; EOFに到達しているか
        AND     EAX,EAX
        JZ      DUMP6           ; EOFでない
        MOV     AH,CLS          ; FILE HANDLEをCLOSE
        MOV     EBX,FHAND
        INT     MSDOS
        IRET                    ; 終了 (P386へ復帰)
;
ERR1:
        MOV     ESI,OFFSET MSG1 ; FILE名の指定なし
        JMP     SHORT ERR0
ERR2:
        PUSH    DS              ; FILE名の表示
        MOV     EAX,GS
        MOV     DS,EAX
        MOV     ESI,FNAM
        CALL    MSGOUT
        POP     DS
        MOV     ESI,OFFSET MSG2 ; OPENできない
        JMP     SHORT ERR0
ERR3:
        PUSH    DS              ; FILE名の表示
        MOV     EAX,GS
        MOV     DS,EAX
        MOV     ESI,FNAM
        CALL    MSGOUT
        POP     DS
        MOV     ESI,OFFSET MSG3 ; READ ERROR発生
ERR0:
        CALL    MSGOUT          ; MESSAGEを表示
        IRET                    ; 終了 (P386へ復帰)
;
NTOH:
        AND     AL,0FH          ; ALの下位NIBBLEを16進数の
        CMP     AL,9            ; 文字へ変換
        JBE     SHORT NTOH1
        ADD     AL,7
NTOH1:
        ADD     AL,'0'
        RET
;
HBOUT:
        PUSH    EAX             ; ALの内容を2桁の16進数で表示
        SHR     AL,4
        CALL    NTOH
        CALL    CO
        POP     EAX
        PUSH    EAX
        CALL    NTOH
        CALL    CO
        POP     EAX
        RET
;
SKPBLN:
        MOV     AL,[ESI]        ; DS:ESIで始まる文字列の最初の
        INC     ESI             ; SPACEあるいはTAB以外の文字を
        CMP     AL,' '          ; ALに、その文字の位置をDS:ESI
        JZ      SHORT SKPBLN    ; に返す
        CMP     AL,09H
        JZ      SHORT SKPBLN
        DEC     ESI
        RET
;
MSGOUT:
        MOV     AL,[ESI]        ; DS:ESIで始まる文字列を表示
        INC     ESI
        AND     AL,AL
        JZ      SHORT MSGOUT1
        CALL    CO
        JMP     SHORT MSGOUT
MSGOUT1:
        RET
;
CO:
        PUSH    EAX             ; ALの1文字を表示
        PUSH    EDX
        MOV     DL,AL
        MOV     AH,COUT
        INT     MSDOS
        POP     EDX
        POP     EAX
        RET
;
        EVEN
ADRS    DD      ?               ; FILE内のDATAのOFFSET
FHAND   DD      ?               ; FILE HANDLE
RLEN    DD      ?               ; DATAの残りBYTE数
EOF     DD      ?               ; FILEのEOFを示すFLAG
MSG1    DB      'FILEを指定してください。'
        DB      CR,LF,0
MSG2    DB      'はOPENできません。'
        DB      CR,LF,0
MSG3    DB      'にREAD ERRORが発生しました。'
        DB      CR,LF,0
;
TEXT    ENDS
;
        END     DUMP
```

リスト3：MAKBIN.BAT

```
MASM %1,%1,NUL/Z/B63/R/MX; > %1.ERR
IF ERRORLEVEL 1 GOTO ERROR
LINK %1,%1; >> %1.ERR
IF ERRORLEVEL 1 GOTO ERROR
EXE2BIN %1.EXE %1.BIN >> %1.ERR
:ERROR
TYPE %1.ERR
```


メモリ管理用語集
デバッグレジスタ　80386以降のCPUでサポートされたブレークポインタを指定するデバッグ用のレジスタ.



特集

286/386メモリ活用プログラミング

プロテクトメモリ管理規約の概要

MS-DOSのメモリ拡張とVCPI, DPMI

菱 博善

MS-DOSからプロテクトメモリを利用する際に，それぞれのアプリケーションが独自の方法でプロテクトメモリを管理していると，さまざまな障害が生じる可能性がある．このような問題を解決するために，プロテクトメモリ管理規約が設けられている．ここでは，80386ベースのパソコンにおいて，EMSエミュレータとDOSエクステンダの共存を実現するプログラムインターフェースであるVCPIと，DOSベースのマルチタスク環境における，プロテクトモードアプリケーションの安全な実行を実現するDPMIについて解説する．

MS-DOSとプロテクトメモリ

MS-DOSを使っていて遭遇する問題点の1つに，メモリ不足があげられます．これは，MS-DOSおよびPC-DOS(以下DOSと表記)が，もともと8088CPUのIBM PCからはじまっていることに起因します．また，標準的なDOSの動作は，たとえCPUが80286，80386，80486になっても，8088，8086とプログラム互換のあるリアルモードに限定されていますので，DOSアプリケーションからアクセス可能なメモリは，リアルモードでアクセス可能なアドレス範囲（原則的に1MB+64KB−16バイト）に限定されてしまいます．

したがって，せっかくPC本体が1MB以上のプロテクトメモリをもっていたとしても，それを直接DOSアプリケーションが使用することは，DOSエクステンダなどを使用しないかぎり困難です．

従来のDOSのメモリ拡張方法としては，LIM-EMS，バンク切り替えなどの方法が考えられ，標準となっているわけですが，それらの特徴や規格などに関しては別項に譲ることにして，本稿では，CPUから見て自然なメモリであるプロテクトメモリと，その管理規約などをみることにします．

プロテクトメモリを使用する上での問題点

DOS上で，何らかの形でプロテクトメモリを使用する場合に，問題となる点としては，プロテクトメモリの管理方法があげられます．IBM PC/ATおよびその互換機では，プロテクトメモリの管理に関して，BIOSレベルである程度の機能がサポートされています(表1)．しかし，これは見ての通り単純にメモリの残量を表すのみであり，メモリに関する基本管理動作である取得・解放・変更などの規定がありません．また，プロテクトモードへの移行に関しても，リアルモード→プロテクトモードの移行はサポートしていますが，逆はサポートされていません．

日本国内で代表的なNECのPC-

表1：IBM PC/ATのBIOS，プロテクトメモリ関連

メモリブロックの移動

```
Int 15H
AH＝87H
CX＝移動するワード数
        最大8000H（最大転送可能メモリサイズ＝64KB）
ES:SI＝GDT へのポインタ
戻り値：
        PC, PCjr                    キャリー／AH=80H
        PC XT, PS/2 Model 30        キャリー／AH=86H
        そのほかの機種
                ノーエラー：
                キャリークリア，ゼロフラグセット
                AH＝0
                エラー：
                キャリーセット，ゼロフラグクリア
                AH=01H          パリティエラー
                   02H          そのほかの例外保護違反
                   03H          A20コントロールエラー
ES:SI の GDT 内容
        00H   ダミー（ゼロにセット）
        08H   GDT セグメントロケーション（BIOS で使用．ゼロにセット）
        10H   ソース側 GDT（ソース側メモリブロックを示すGDT）
        18H   デスティネーション側GDT（デスティネーション側メモリブロックを示すGDT）
        20H   BIOS コードセグメント（BIOS で使用．ゼロにセット）
        28H   BIOS スタックセグメント（BIOS で使用．ゼロにセット）
```

プロテクトモードメモリサイズの取得

```
Int 15H
AH＝88H
戻り値：
        PC, PCjr                    キャリー／AH=80H
        PC XT, PS/2 Model 30        キャリー／AH=86H
        そのほかの機種
        AX＝1MB 以上に使用可能な連続したメモリのサイズ（1KB 単位）
```

プロテクトモードへの移行

```
Int 15H
AH＝89H
BL＝IRQ0割り込み番号
BH＝IRQ8割り込み番号
CX＝プロテクトモードの制御移行アドレス（オフセット）
ES:SI＝GDT へのポインタ
戻り値：
        キャリーセット＝エラー
        キャリークリア＝ノーエラー
```


メモリ管理用語集
動的システム　ソフトウェアのサイズが実装メモリよりも大きく，システムを常駐部分と非常駐部分に分けて実行するもの．

9801/98シリーズにおいては，公式にはプロテクトメモリの管理方法などが公表されていないようです．もっとも，筆者の不勉強なのかもしれませんが．

実際問題として，もしプロテクトメモリを使用するプログラムが１つのみで，いったん起動したあとには，電源断（またはリセット）まで絶対に終了しない，というのであれば，たとえプロテクトメモリに関するシステム上の規約がいっさいなくても，何ら問題はないと考えられます．しかし，現実にはプロテクトメモリは，DOSのRAMディスクなどとしても使用されている(かもしれない)ために，勝手にアクセスするだけでよいというものでもありません．また，80386以上のCPUで仮想86モードEMSなどが動作していた場合には，その中からプロテクトモードへ移行することは事実上不可能です．

このような中で，必然的に発生してきた規格としてXMS（eXtended Memory Specification）があります．筆者の認識によると，もともとこの規格はWindowsの動作上，先に記したような管理上の問題を解決するために決められたようですが，規格自体としてはあまり一般的に知られているとはいえないようにも思えます．

その次に決められた規格としてはVCPIがあります．この規格自体は，80386の仮想86モードEMSドライバの環境下からプロテクトメモリを直接使用するためのもので，80386以上のCPUでのみ使用可能な規格となっています．この規格によって，はじめて仮想86モードEMSとプロテクトモードのプログラム動作が共存できるようになったといえます．さらに，昨年発表されたDPMI規格は，その名称の通り，MS-DOSのメモリ管理などをプロテクトモードにまで拡張するものとなっています．ただし，現時点ではWindows 3.0の内部的に使用される規格として捉えることができます．以下では，このVCPIとDPMIに関して見てみることにします．

VCPI

VCPIとは何か

VCPIは，Virtual Control Program Interfaceの略です．これは1987〜1988年にかけて，DOSエクステンダで有名なPhar Lap Software，DESQView/QEMM386（仮想86モードEMSドライバ）のQuaterdeck，386MAX（仮想86モードEMSドライバ）のQualitas，およびA.I.Archtects，Quadram，RationalSystemsの６社によって，複数の80386プログラムが同時に実行されるときに発生する問題を解決するために決められました．以降，各種のDOSエクステンダによりサポートされ，現在ではIBM PC/ATの世界で事実上の標準になっています．

VCPI自体は，規格そのものでプログラムではありません．しかし，規格自体が80386CPUで，仮想86モードEMSドライバを前提として決められていますので，実質的には仮想86 EMSドライバのための機能拡張と捉えることもできます．事実，規格自体のVCPIファンクションコールはEMSのファンクションコールと同じくInt 67Hを通して行います．また，VCPIをサポートしている環境で，仮想86モードEMSドライバの特性をまったく含まないものは存在しないと考えられます．

現実に存在する仮想86モードEMSドライバでVCPIをサポートしているものとしては，IBM PC/ATでは，QEMM386，386MAX，DOS Ver 5.0のEMM386などがあげられます．

VCPIのファンクションと機能

VCPIを機能面から見ると，実に単純なものばかりとなっています（表2）．大別すると，メモリの取得と解放に関与するもの，仮想86モード内からプロテクトモードへの制御移行・復帰に関与するものの２種類に分けることができます．それ以外の，GDT，IDTなどの管理に関する規約などはいっさい決められていません．もちろん，割り込みのハンドリングなどに関する規約もありません．

したがって，VCPIを使用するプログラムは，プロテクトモードに関与する情報，GDT，LDT，IDT，CR3，ページテーブルなどの管理をすべて自前でもち，コントロールしなければなりません．これでは，いったい何のためにある規格なのかわからないような気もします．しかし，逆に考えると，プログラム上規格による制約がほとんどないので，CPUの資源をすべて自分のものとして使用することもできます．その意味では，VCPIは，仮想86モードEMSから，CPUの制御権を取り戻すための規格と考えられなくもありません．

メモリ管理用語集
DOSエクステンダ　MS-DOSから1MBを超えるメモリ空間を利用可能にする一種のドライバ．

ユーザーにとっての意味

先に見てきたように，VCPI規格は極めて単純なものです．また，VCPIを直接使うためには，80386の機能・制約などを熟知している必要があります．このことから，ユーザーがVCPIのファンクションコールを直接使うには，よほどのパワープログラマでもないかぎりかなり困難であると考えられます．むしろ，VCPIはその生い立ちからして，DOSエクステンダを仮想86EMSの中で動かすための規格と割り切ったほうがよいでしょう．

一般に，DOSエクステンダを使用したプログラムを使用する場合には，DOSエクステンダの存在，その背後のVCPI，ハードウェア構成はユーザーの目に直接触れません．また，プログラムを作成する場合でも，特定のDOSエクステンダで開発を進めるかぎり，DOSエクステンダの背後にあるVCPI，ハードウェア構成は，原則的に意識されるものではないと考えられます．

表2：VCPIのファンクション概要

表2-1：リアルモードインターフェース
リアルモード(仮想86モード)でのVCPIインターフェースは，EMSの割り込みとおなじInt 67Hを通して行われる．VCPIのファンクションは，EMSドライバの常駐を確認したあとに，AH=0DEHでALにサブファンクションを入れてInt 67Hを実行する．通常，VCPIをサポートしていないEMSドライバでは，AH=0DEHはサポートされていないファンクションなので，サポートされていないファンクションに対するエラーとなり，VCPIの存在が確認できる．

サブファンクション	内容
00H	VCPIの存在チェック
01H	プロテクトモードインターフェースの取得
02H	最大物理メモリアドレスの取得
03H	4KB単位の未使用ページ数取得
04H	4KBのメモリを割り当て
05H	4KBのメモリを解放
06H	0～1MB内の4KBページの物理アドレスを取得
07H	CR0の読み出し
08H	デバッグレジスタの読み出し
09H	デバッグレジスタの設定
0AH	8259A割り込みコントローラのベクタを取得
0BH	8259A割り込みコントローラのベクタの設定
0CH	プロテクトモードへの制御移行

表2-2：プロテクトモードインターフェース
プロテクトモードインターフェースは，リアルモードインターフェースのサブファンクション01H／プロテクトモードインターフェースの取得で得られたセレクタ：オフセットに対して，Use32セグメントに対するファーコールを行うことによって実行される．このとき，AH=0DEHとする．AL=サブファンクション．

サブファンクション	内容
03H	4KB単位の未使用ページ数取得
04H	4KBのメモリを割り当て
05H	4KBのメモリを解放
0CH	リアルモード（仮想86モード）への制御移行

DPMI

DPMIとは何か

DPMIとはDos Protected Mode Interfaceの略で，文字通りDOS上でプロテクトモードプログラムを実現するための規格です．DPMIは先にも述べたように，Windows 3.0の内部規格としてサポートされています．現時点では，筆者の知るかぎりでは，Windows 3.0の外部でDPMIが実現されている環境はありません．規格の発生元は，いうまでもなくマイクロソフトです．当初はいろいろあったのでしょうが，現在ではもともとVCPIをサポートしていたDOSエクステンダのメーカーもDPMIをサポートし始めているようです．

DPMIは，本来の規格の目的はともかく，現実にはWindows 3.0に内蔵されるものであり，Windows 3.0自体はCPUが何であれ動作するものです．このためか，DPMIは80386/80486のみならず，80286CPUであっても使用可能なものとなっています．もちろん，80286ではページングなどの80386の機能が使用できないのはいうまでもありません．

メモリ管理用語集
トラップ　命令を実行したあと処理をする例外処理．

DPMIのファンクションと機能

DPMIのファンクションは多岐に渡り複雑なものとなっていますので，本稿では一部を紹介するのみとしました（表3）．DPMIを機能面から見ると，実はDPMI自体がDOSエクステンダの機能を一部含んでいることに気がつきます．DOS Protected Mode Interfaceという名称にも，そのことが表れているようです．

DPMIがWindows 3.0の中で実現されていることから，DPMI自体はWindows 3.0内部のメモリ・割り込みなどの管理機構の一部と捉えられなくもありません．また，現在公表されているDPMIのファンクションが，DOSエクステンダとして十分なものとはいえないような気もします．しかし，Windows 3.0の中では，明らかにDOSエクステンダに相当する動作環境が存在しているわけですから，ひょっとするとただ単に公開されていないだけで，実際には非公開のDPMIファンクションとして，いろいろな機能が備わっているのかもしれません？

ユーザーにとってのDPMI

少なくとも，ユーザーがWindowsとかDOSエクステンダを使用しているかぎりでは，DPMIの存在を意識することはないでしょう．この理由は，VCPIと同じく，たとえユーザーの接するアプリケーションの背後でDPMIが存在し動いていたとしても，ユーザーの意識するところにはならないということです．同様に，プログラマのレベルでも，高級言語でプログラムするかぎり，たとえその環境が何であれ，DPMIに直接接する機会は少ないものと考えられます．また，Windows 3.0のプログラムを書くということ自体が，間接的にDPMIを使用するプログラムを書いているともいえるでしょう．

表3：DPMI 1.0のファンクション概要

表3-1：DPMIのファンクションアクセス方法(等)

DPMIを使用するには，

1．DPMIのインストールをチェックする
2．インストールチェックで取得したアドレスへファーコールをかけ，プロテクトモードへ制御を移行する
3．プロテクトモードへ移行したあとは，Int31HによりDPMIファンクションを実行する．DPMIのファンクション呼び出しは，INT31Hによって行われ，DPMI下でプロテクトモードになっている場合に有効．
4．プログラムを終了するときは，Int21H／AH=4CHにより終了する
Int21H／AH=4CHの実行により，プログラムは終了し，制御はリアルモード（仮想86モード）へ移行する．これ以外のDOSの終了ファンクションは，DPMIではサポートされていないので，使用してはならない．

DPMIのインストールチェック

```
Int  2FH
AX=1687H
出力
AX：0=ノーエラー        !0=エラー
BX：bit0=1であれば32ビットプログラムがサポートされている
CL：プロセッサタイプ
        02H     80286
        03H     80386
        04H     80486
DH ：DPMIメジャーバージョン
DL ：DPMIマイナーバージョン
SI ：DPMIホストのプライベートデータに必要な領域のパラグラフ数（おそらく0）
ES：DI：プロテクトモード移行ルーチンのエントリアドレス
```

プロテクトモード移行ルーチンエントリの呼び出し方

```
AX：プログラムが32ビットアプリケーションのときbit0=1
ES：DPMIホストのプライベートデータに必要な領域のパラグラフ数
    0のときは無視される．
出力：
成功したとき＝キャリークリア
          プログラムはプロテクトモードで実行されている
    CS＝リアルモードのCSをベースとした64KBリミットの16ビットセレクタ
    SS＝リアルモードのSSをベースとした64KBリミットのセレクタ
    DS＝リアルモードのDSをベースとした64KBリミットのセレクタ
    ES＝プログラムのPSPをベースとした100バイトリミットのセレクタ
    FS／GS＝0（80386/486のときのみ）
    もしプログラムが32ビットアプリケーションの場合はESPの上位ワードは0
失敗したとき＝キャリーセット
          プログラムはリアルモードで実行継続
```

DPMIファンクションの呼び出し方

```
INT  31H
AX＝ファンクション番号
そのほかのレジスタは，ファンクションにより設定が必要．
```

VCPIとDPMIの比較

VCPIとDPMIを比較してみると，目的とする機能が根本的に異なっていることに気がつきます．VCPIは，仮想86EMSとDOSエクステンダなどが共存することを目的としています．一方のDPMIは，それ自身がDOSエクステンダ的な機能を取り込み，Windows 3.0の中でDOSのプログラムをそのまま実行する仮想マシン機能を提供し（DPMIの中で別のDOSエクステンダが動くかどうかは別としても），DOS自身の機能を強化することを目的としていることは明らかです．

では，それぞれでできることはどうでしょう？　VCPIは，ほとんど100%プログラマ任せですので，CPU資

表3-2：DPMIのファンクション概要

ファンクション番号	内容
LDT管理ファンクション	
0000H	LDTディスクリプタの割り当て
0001H	LDTディスクリプタの解放
0002H	リアルモードのセグメントをディスクリプタに割り付ける
0003H	次のセレクタへの増加値を取得
0004H	予約
0005H	予約
0006H	セグメントベースアドレスの取得
0007H	セグメントベースアドレスの設定
0008H	セグメントリミットの設定
0009H	ディスクリプタアクセス権の取得
000AH	コードセグメントエイリアスのディスクリプタの作成
000BH	ディスクリプタの取得
000CH	ディスクリプタの解放
000DH	指定のLDTディスクリプタの割り当て
000EH	複数ディスクリプタの取得
000FH	複数ディスクリプタの解放
DOSメモリ管理ファンクション	
0100H	DOSメモリブロックの割り当て
0101H	DOSメモリブロックの解放
0102H	DOSメモリブロックのサイズ変更
拡張メモリ管理ファンクション	
0500H	未使用メモリ情報の取得
0501H	メモリブロックの割り当て
0502H	メモリブロックの解放
0503H	メモリブロックのサイズ変更
0504H	連続したメモリブロックの割り当て
0505H	連続したメモリブロックのサイズ変更
0506H	ページ属性の取得
0507H	ページ属性の変更
0508H	デバイスをメモリブロック内にマップ
0509H	0～1MBのメモリ（コンベンショナルメモリ）をメモリブロック内にマップ
050AH	メモリブロックのサイズ取得
0800H	物理アドレスのマッピング
0801H	物理アドレスのマッピング解放
0D00H	共有メモリの割り当て
0D01H	共有メモリの解放
0D02H	共有メモリのシリアライズ
0D03H	共有メモリのシリアライズの解除
ページ管理ファンクション	
0600H	連続した領域のロック
0601H	連続した領域のロック解除
0602H	リアルモード領域をページング可能としてマークする
0603H	リアルモード領域の再ロック
0604H	ページサイズの取得
0700H	予約
0701H	予約
0702H	ページをデマンドページングの候補としてマークする
0703H	ページ内容を破棄する
割り込み管理ファンクション	
0200H	リアルモード割り込みベクタの取得
0201H	リアルモード割り込みベクタの設定
0202H	プロセッサ例外ハンドラベクタの取得
0203H	プロセッサ例外ハンドラベクタの設定
0204H	プロテクトモード割り込みベクタの取得
0205H	プロテクトモード割り込みベクタの設定
0210H	プロテクトモード拡張プロセッサ例外ハンドラベクタの取得
0211H	リアルモード拡張プロセッサ例外ハンドラベクタの取得
0212H	プロテクトモード拡張プロセッサ例外ハンドラベクタの設定
0213H	リアルモード拡張プロセッサ例外ハンドラベクタの設定
0900H	仮想割り込み状態の取得と仮想割り込み禁止
0901H	仮想割り込み状態の取得と仮想割り込み許可
0902H	仮想割り込み状態の取得
翻訳ファンクション	
0300H	リアルモード割り込みのシミュレート
0301H	ファーリターンフレームによるリアルモードプロシージャの呼び出し
0302H	割り込みリターンフレームによるリアルモードプロシージャの呼び出し
0303H	リアルモードコールバックアドレスの割り当て
0304H	リアルモードコールバックアドレスの解放
0305H	状態セーブ（保存／復元）アドレスの取得
0306H	直接CPUモード切り替えアドレスの取得
そのほかのファンクション	
0400H	DPMIバージョンの取得
0401H	DPMI可能ファンクション範囲の取得
0A00H	ベンダー個別APIエントリポイントの取得
0B00H	デバッグウォッチポイントの設定
0B01H	デバッグウォッチポイントのクリア
0B02H	デバッグウォッチポイントの状態取得
0B03H	デバッグウォッチポイントのリセット
0C00H	常駐ハンドラ初期化コールバックのインストール
0C01H	常駐終了
0E00H	コプロセッサの状態取得
0E01H	エミュレーションの設定

メモリ管理用語集
バインダ　動的システムの開発でオブジェクトファイルを結合するユーティリティ．

源に可能なことはほぼ100％可能であると考えられます．いい換えれば，VCPI環境下でできないことはないといえるでしょう．ここで注意して欲しいのは，ある機能ができないということと，サポートされていないということはまったく別であるということです．

一方DPMIのほうは，いたれりつくせりの何でもありに近く，DOS環境と互換性のあるプロテクトモードプログラムにとって，必要なことはかなり揃っているように見えます．逆にDPMIでは，CPUの保護機能が生かされているので，何でもできるというわけではなく，ある意味では制限事項が多いといえなくもありません．

このことは，プログラムを作る側から見た場合，プログラムの負担として表れてきます．もしVCPIを直接扱うプログラムを書こうとしたら，80386の基本的な知識，プロテクトモード，各種保護機能，ページング関連機能，マルチタスクサポート機能など，ほとんどあらゆることを意識し，すべての処理をプログラム側で行わなければなりません．それに比してDPMIの場合には，ある程度の機能はあらかじめ揃っているので，DPMIを直接扱う場合でも（ファンクションの全容が明確になっていれば）さほど難しくないでしょう．

ただ，いずれの場合でも共通して考えられるのは，これらを直接アクセスするプログラムを書こうとする場合には，程度の差こそあれ，80286/80386のプロテクトモードに関する知識が必要になるということです．

アプリケーション面ではどうでしょうか？　VCPIは歴史が古い（といってもたいしたことはないが）せいもあり，IBM PC/ATの世界では事実上の標準となっています．代表的な仮想86EMSドライバとDOSエクステンダでは，ほとんどVCPIがサポートされています．もっとも，これらのメーカーは，ほとんどの場合VCPIを最初に定めたメンバーに含まれていますので，当然といえば当然ともいえます．

一方のDPMIは，Windowsアプリケーション如何ではないかという感じがします．また，DPMIは，まだ規格自体の全容が明らかになっていないようです．たとえば，Windows 3.0は外部のDPMI環境で動作するのかどうか？　といったような問題が明らかになっていません．これらも含めて，考えていかなければならない項目が（ファンクションとおなじように）多義にわたって存在しています．

将来はどうなるか？

以上，ちょっと古い規格であるVCPIと，ちょっと新しい規格であるDPMIを見てきたわけですが，今後これらの規格の占める位置とその普及はどうなるでしょうか？　VCPI自体は，すでに標準となっています．事実，新しいDOS 5.0のEMM386では，QEMM386などと同様にVCPIがサポートされています．これから見ても，今後も，少なくともNon-Windowsアプリケーションの世界ではVCPIがなくなることはないでしょう．

DPMIのほうは，まだよくわかっていない部分も多いわけですが，少なくともWindows 3.0にDPMIが付属しているわけですから，Windows 3.0の普及に伴って，意識するしないにかかわらず普及するものと思われます．しかし，ほんとうの意味でDPMIがDOSのプロテクトモードインターフェースといえるようになるには，DOS自身が何らかの方法でDPMIをサポートするようにならなければならないのではないでしょうか？

参考文献：
VCPI　Version1.0　June 12, 1989
BYTE IBM Special Edition Fall 1989 / Storetching Dos to the Limit
PC MAGAZINE FEBRUARY 12, 1991 / Power Programing, An Introduction to The DOS Protected Mode Interface
DOS PROGRAMMER'S REFERENCE / QUE

バンクメモリ　メインメモリ80000Hから9FFFFHまでの128KBをポートをアクセスすることによってメモリページを切り替える拡張方式

特集

286/386メモリ活用プログラミング

80286，80386のプロテクトモードを使う

DOSエクステンダ活用プログラミング

株式会社LIFEBOAT　森 誠

DOSエクステンダとは，MS-DOSのもつ640KBのバリヤーを破り，プロテクトモードを利用してメモリ不足を解決するためのプログラミングインターフェースとソフトウェアプロダクトのことである．DOSエクステンダがMS-DOSとプロテクトモードで動作するアプリケーションとのインターフェースをとってくれるので，プログラマはMS-DOS上でプロテクトモードを利用したアプリケーションプログラムを開発することが可能になる．このDOSエクステンダとDOSエクステンダを利用したプログラミングの実際について解説を行った．

DOSエクステンダの概要

DOSエクステンダとは，プロテクトモードでプログラムを実行するためのツールの総称です．XMS (eXtended Memory Specification) が，プロテクトモードメモリにデータ領域を確保するためのデバイスドライバの仕様であるのに対し，DOSエクステンダはプログラム自体をプロテクトモードのまま実行することを目的としています．また，EMS (Expanded Memory Specification)やXMSではその機能を実現するのは，通常EMM(Expanded Memory Manager) や XMM (eXtended Memory Manager) などのデバイスドライバですが，ほとんどのDOSエクステンダはアプリケーションプログラムに結合されます（一部には，DOSエクステンダのカーネルがTSR形式になっていて常駐するものもありますが，これは例外的存在です）．

DOSエクステンダの起源

現在，市販されているDOSエクステンダの多くは，アプリケーション開発の副産物が製品化されたものです．これらの中には，開発したアプリケーションのプログラム領域を拡張するために作られたものもありますが，本来の目的はメモリの拡張ではなく，80386ボード用プログラムをDOSからクロス開発するときに，プロトタイプをDOS上で動作させるためのツールや，パソコン用の拡張386ボードのドライバプログラムだったものを，DOSエクステンダとして商品化したものです．そのため，同じDOSエクステンダといっても，プログラムの特権レベルやメモリマップ，システムセグメントが違ったりします．

開発元の意図とは別に，ユーザーにはそのソフトウェアをDOSのメモリ拡張に使いたいという欲求があり，それぞれDOSエクステンダへと発展していきました．DOSエクステンダは，ユーザーのニーズに基づいて各製品が独自に発展したため，現状ではアプリケーションプログラムとDOSエクステンダとのインターフェースが製品ごとに異なったものとなっています．

DOSエクステンダの種類

DOSエクステンダの中には，80286のプロテクトモードを使用したものと80386のプロテクトモードを使用したものがあります．

80286DOSエクステンダは，設置ベースでいまもっとも多くのパソコンで使われている80286で動作し，16MBまでメモリを拡張できます．さらに，80286DOSエクステンダは80386以上のCPUでも使用できます．しかし，80286のプロテクトモードは16ビットで，セグメントの大きさは最大64KB，つまり8086同様，セグメント境界が存在するわけです．このため，セグメントにまたがるデータの処理は80386ほど簡単でなく，速度は80386DOSエクステンダほど劇的には上がりません．80286DOSエクステンダには，RasionalSystems社のDOS/16M，ErgoComputing社のOS/286などがあります．

80386のプロテクトモードを使用した80386DOSエクステンダでは，80386のさまざまな機能が使えるため，アプリケーションプログラムに多大のメリットを与えることができます．ただし，80286マシンでは使用できなくなります．80386DOSエクステンダには，RationalSystems社のDOS/4G，PharLapSoftware社の386DOS|Extender，ErgoComputing社のOS/386などがあり，これらは80386／80386SX／80486で動作可能です．なお，80386SX，80386DX，80486SX，80486DXは，ソフトウェア的には（つまりプログラムから見ると）違いがないので，以降これらのプロセッサを80386と総称します（80386SXと80386DXとの違いは，外部バスが80386SXでは16ビット，80386DXでは32ビットである点です．また，80486DXには数値演算コプロセッサとキャッシュメモリが内蔵され，いくらかのマルチプロセッサ対応の命令が追加されています．80486SXは，80486DXから数値演算コプロセッサを省略したものです）．

80286では物理アドレスは最大16MB，80386では最大4GBで，DOSエクステンダはこれらのメモリ空間をフルに利用できます．また，DOSエクステンダはMS-DOS上で動作するため，特殊なOSへ移行することなくプロテクトモードの大きなメモリ空間のメリットを享受でき，さらに従来のメモリ常駐型プログラムと共存することができます．

ただし，80386の仮想86モードを使用しているデバイスドライバや常駐型プログラムの上では実行することがで

メモリ管理用語集
ハンドル　EMSメモリをアプリケーションから利用する場合，メモリ取得時にハンドルと呼ばれる識別番号を使って読み書きする．

きません．それは，仮想86モードはプロテクトモードの特権レベルが低く，プロテクトモードへ移行する命令が実行できないからです．もしこの命令を使おうとすると，仮想86モードの環境を作ったプログラムに制御が移ってしまいます．このため，仮想86モードからプロテクトモードへの移行を可能にするには，仮想86モードの環境を作ったプログラム（仮想86モードを利用しているデバイスドライバ）が，仮想86モードからプロテクトモードにモードを変更する仕組を用意している必要があります．米国では，そのための統一規格としてVCPIとDPMIが決められています．DOS/4G，DOS/16M，386｜DOS Extender Ver.4はこのDPMIに準拠しているので，この規格にあったデバイスドライバ上でこれらのDOSエクステンダを動作させることができます．

DOSエクステンダを使ったプログラムは，MS-DOSファンクションコールが使えるので，標準のCのライブラリのほとんどを利用できます．また，プロテクトモードでは，論理アドレスは，ディスクリプタテーブルやページディレクトリ，ページテーブルで物理アドレスに変換されるので，リアルモードのような絶対番地アクセスはできません．しかし，DOS/4GとDOS/16Mでは，VRAMに対してリアルモードと同じアドレスを使って直接アクセスできるように工夫されています．386｜DOS Extenderの場合もVRAM直接アクセス用の特別なセグメントを用意していますから，リアルモードプログラムから移植する場合には，そのベースセグメントの値を変更するだけで行えます．

プロテクトモード概要

ここで，80286，80386のプロテクトモードについて説明します．プロテクトモードでは，リアルモードに対して次の機能が加わります．1つは複数のタスクを制御する機能，もう1つは論理メモリ空間によるメモリ管理および仮想記憶，そして80386では，さらに32ビットリニアアドレス空間が加わります．これらは，プロテクトモードのときのみ機能します．

メモリ管理の機能を使用して，80286では実装メモリ16MB（論理空間1GB），80386では実装メモリ4GB（論理空間64TB）までのメモリをアクセスすることが可能になっています．DOSエクステンダは，通常，仮想記憶（ディスクを使って，実装されているメモリ以上の大きさのメモリがあるかのようにプログラムを実行させる方法）をサポートしていませんが，仮想記憶を利用するためのオプション製品があります．仮想記憶をサポートするオプション製品を追加して使用すると，ディスクをメモリ領域の代わりに使用できますから，たとえば，DOSメモリ640KB+プロテクトモード対応メモリ1MBのパソコンでも，1.6MBを越えるプログラムを動作させることができます．

プロテクトモードでは，セグメントレジスタはセグメントセレクタと呼ばれ，メモリ上のテーブルを指します．そのテーブルには，リニアアドレスのベースアドレスやそのセグメントの大きさ，特権レベル，コードセグメントかデータセグメントかなどの属性などが記述されています．したがって，リアルモードで物理アドレスを得るために行えた，「オフセットの値を4ビットずらしてセグメントレジスタに加える」というような計算はプロテクトモードではできなくなります．そのかわり，80286プロテクトモードでは，8086と同じセグメントオフセット形式でより大きなアドレス空間をアクセスできるようになっています．

一方，80386の場合には，1つのセグメントを32ビットのオフセットでアクセスできるため最大4GBまでアクセスすることができます．そこで，80386DOSエクステンダでは，基本的にセグメントを固定し，オフセットの

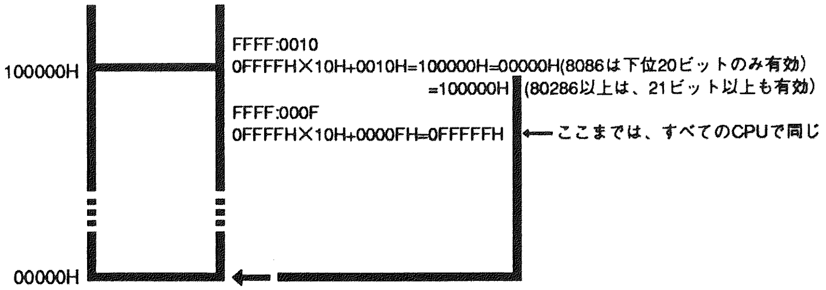
図1

```
「８０８６アドレス計算」
        add     si, INCREMENT_SIZE
        jnc     SKIP
        mov     ax, ds
        add     ax, 1000h
        mov     ds, ax
SKIP:
「８０３８６アドレス計算」
        add     esi, INCREMENT_SIZE
```


メモリ管理用語集
BMS（Bank Memory Specification）　アイ・オー・データ機器が提唱するIOバンク方式のメモリをアクセスするための仕様

図2

8086と80286以上のCPUによるアドレッシングの相違



みでアドレス計算を処理します．これを80386のフラットモデルと呼びますが，これはちょうど8086のCOMモデルのオフセットの大きさが16ビットから32ビットに変わったようなものです．8086や80286プロテクトモードでは，64KBを越えるデータはコンパイラが疑似的に連続アクセスできるように，オフセットのオーバーフローを検出しながらセグメントアドレスを計算していました．80386フラットモデルでは，このオフセットオーバーフロー検出・セグメント計算がいっさい不用になるので大量データを効率的に処理できます(図1参照)．

80286には，リアルモードからプロテクトモードへ移る命令はありますが，プロテクトモードからリアルモードへ戻るためにはCPUにリセットをかけなければなりません．80286を搭載しているたいていのパソコンには，CPUにリセットをかける機構と，リセット時に電源が入った状態とプロテクトモードから戻ってきた状態を区別する仕組が組み込まれています．80286DOSエクステンダでは，この仕組を使ってプロテクトモードのアプリケーションが終了したときなどに，リアルモードに戻ってDOSに制御を返します．この仕組はCPUの機能ではないので，機種によってその方法が違います．このため，80286DOSエクステンダには機種依存性があります．

80386には，リアルモードからプロテクトモードへ，プロテクトモードからリアルモードへと双方向に移る命令があるので，モード切り替えに関しては機種依存性がありません．ただし，アドレス線の制御に関する問題で機種依存性が生じます．8086のアドレスバスが20本なのに対して，80286では24本，80386では32本のアドレスバスが出ています．このため，リアルモードの時でさえ，FFFF:000Fを越えるアドレスについて，8086が0番地アクセスになるの対して，80286以上のCPUでは100000H番地アクセスになるという違いがあります(図2参照)．

21本目のアドレス線であるA20ラインを無効にすれば，80286以上のCPUでも8086と同じ動作になります．そこで，多くの80286／80386パソコンでは，このA20ラインの有効／無効を制御する仕組を組み込んでいて，それをソフトウェアで制御できるようにしています．A20ラインの制御は80386CPUの機能ではないので，その機能は機種に依存し，機種ごとに実現方法が異なっています．このため，機種別に対応した80386DOSエクステンダが必要になりました．

DOSエクステンダの動作原理概要

DOSエクステンダで作られたプログラムはプロテクトモードで動作しますが，MS-DOSはリアルモードで動いています．そこで，DOSエクステンダのパッケージには，リアルモードとプロテクトモードの橋渡しをするプログラムが入っています．アプリケーションプログラムに起動がかかると，まず，そのプロテクトモードとリアルモードの橋渡しをするプログラムが動作します．このプログラムはCPUをリアルモードからプロテクトモードに移し，DOSエクステンダのシステムルーチンの一部とアプリケーションプログラムをメモリにロードします(図3参照)．

このとき，1MB以上のアドレスの拡張メモリを先に使うDOSエクステンダと1MB以下のDOSメモリを先に使うDOSエクステンダとがあります．拡張メモリを先に使うDOSエクステンダでも，拡張メモリにアプリケーションをロードするだけの大きさがなければDOSメモリを使ってロードします．この違いは，DOSエクステンダの基本設計において，子プロセスなどでリアルモードのDOSアプリケーションを起動したり，DOSエクステンダを使ったTSRプログラムを作成することを意識しているかどうかの違いです．

ロード終了後，制御はアプリケーションに渡されます．プログラム実行中にMS-DOSファンクションコールが起きると，いったん制御はDOSエクステンダに移り，リアルモードに戻ってからMS-DOSを呼び出します．ファンクションコールが終了すると，再びプロテクトモードに移行してアプリケーションに制御を戻します．アプリケーションが終了すると，制御はDOSエクステンダのシステムに戻ってきますので，そこでプロテクトモードからリアルモードに戻り，制御をMS-DOSに返します(図4参照)．

図4
リアルモード／プロテクトモードの切り替え

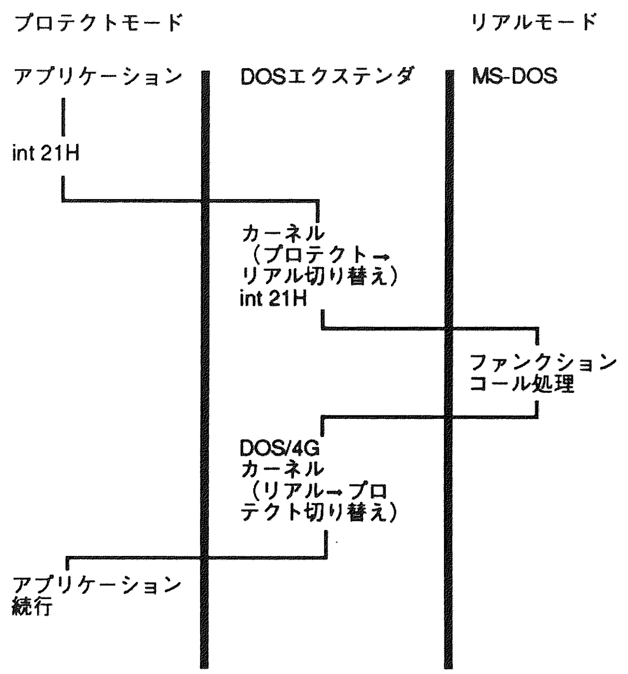
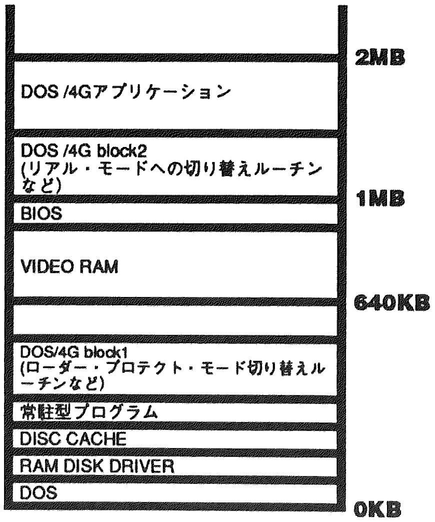


図3
メモリ配置図(DOS/4Gの場合)



DOSエクステンダの開発環境

DOSエクステンダの開発言語およびツールを説明します．開発言語として80286DOSエクステンダの場合には，Microsoft CやLattice Cなど，通常のリアルモード開発用のCコンパイラが使用できます．これは，リアル

メモリ管理用語集
VCPI　DOSエクステンダとEMSエミュレータを共存させるための規格．1987年米PharLap社をはじめとする6社によって提唱された．

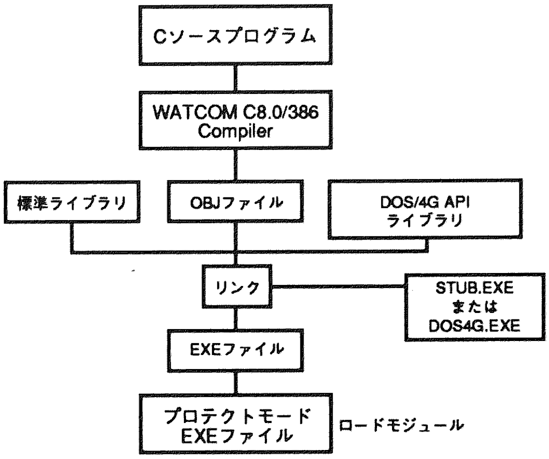
モードも80286プロテクトモードも，16ビットセグメントと16ビットオフセットの組み合わせというアドレス形式が同じであるため，リロケータブルオブジェクトのレベルでの変更がなく，リアルモード用のコンパイラが流用できるからです．しかし，80386DOSエクステンダの開発には，80386に特化した32ビットフラットモデルの機能を十分に引き出す専用コンパイラが必要になります．米国では，WATCOM C/386やHigh C/386などが使われています．

コンパイラの出力したオブジェクトを実行形式のプログラムにするためにリンカが使われますが，この実行形式は通常のMS-DOSの実行形式ではなく，プロテクトモードの実行形式となります．ですから，DOSエクステンダの専用リンカを使ったり，80286DOSエクステンダの場合には，リンカはDOSのリンカを使って，それをプロテクトモードの実行形式に変換するツールがDOSエクステンダとともに提供されていたりします(図5参照)．

開発中のデバッグに関するツールですが，プロテクトモードで実行されるプログラムをデバッグするのですから，当然DOS用のデバッガを使うことはできません．DOSエクステンダに付いているデバッガは，製品によってかなりレベルが異なります．DOS/16MやDOS/4GのようにInstant-Dというソースコードウィンドウ指向デバッガがバンドルされている製品から，単なるシンボリックデバッガが添付されているものまであります．

DOSエクステンダにはDOSエクステンダの本体であるカーネル(ローダ)と，カーネルをアプリケーションプログラムと結合するバインダが入っています．バインダを使ってカーネルをリンカが出力した実行形式ファイルと結合すると，通常のDOSのアプリケーションと同じように使えるDOSエクステンダのアプリケーションができあがるわけです．

図5 アプリケーション開発行程(DOS/4Gの場合)



開発手順例

MS-DOSのリアルモードプログラムの開発手順との違いは，80286DOSエクステンダの場合には，リンカ・デバッガがDOSエクステンダ専用になるのとライブラリに若干の差し替えが起こることです．80386DOSエクステンダの場合には，さらにコンパイラも80386プロテクトモード専用のものになります．開発途中では，カーネルをアプリケーションに組み込む必要はありませんが，デバッグ完了後にカーネルをアプリケーションに組み込んで，最終的なプログラムに仕上げます．

例として，DOS/4Gにおけるプログラム開発を取り上げます．

ここで，リスト1およびリスト2のようなプログラムを作成したとします．リスト1は引数で与えられた位置に，指定の大きさのキャラクタモードのウィンドウを開き，ファイルの内容を表示するプログラムです．リスト2はそのメインプログラムで，コマンドラインを調べてウィンドウの位置，大きさ，表示するファイル名を取り込むプログラムです．プログラムができたら，次のようにコンパイルします．

```
wcl386 /zk0 /c test1.c test2.c
```

次に，80386専用リンカGLU32(General Linking Utility)を使ってリンクします．

リスト1：TEST1.Cキャラクタウィンドウプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#define MAX_WIDTH 80
static unsigned int *savescreen;
static int pos_x, pos_y, pos_width, pos_raw;
static unsigned *VRAM = (unsigned *)0xa0000;
openwindow(x, y, width, raw, fp)
int x, y, width, raw;
FILE *fp;
{
unsigned int *scrptr, *bufptr;
int i, j, flag, c;

pos_x = x;
pos_y = y;
pos_width = width;
pos_raw = raw;
if(!(bufptr = savescreen = halloc((long)width * raw, sizeof(unsigned int)))){
        printf("halloc error¥n");
        exit(3);
        }
for(scrptr = VRAM + MAX_WIDTH * y + x, i = 0; i < raw; i++) {
        for(j = 0; j < width; j++)
                *bufptr++ = *scrptr++;
        scrptr += MAX_WIDTH - width;
        }
scrptr = VRAM + MAX_WIDTH * y + x;
*scrptr++ = 0x98; /* ┌ */
for(j = 1; j < width-1; j++)
        *scrptr++ = 0x95; /* ─ */
*scrptr++ = 0x99; /* ┐ */
scrptr += MAX_WIDTH - width;
for(i = 1; i < raw - 1; i++) {
        *scrptr++ = 0x96; /* │ */
        for(j = 1, flag = 0; j < width - 1; j++)
                if(flag || (c = getc(fp)) == EOF || c == '¥n') {
                        *scrptr++ = ' ';
                        flag = 1;
                        }
                else *scrptr++ = c;
        *scrptr++ = 0x96; /* │ */
        scrptr += MAX_WIDTH - width;
        }
*scrptr++ = 0x9a; /* └ */
for(j = 1; j < width-1; j++)
        *scrptr++ = 0x95; /* ─ */
*scrptr++ = 0x9b; /* ┘ */
}
closewindow()
{
unsigned int *scrptr, *bufptr = savescreen;
int i, j;
for(scrptr = VRAM + MAX_WIDTH * pos_y + pos_x, i = 0; i < pos_raw; i++) {
        for(j = 0; j < pos_width; j++)
                *scrptr++ = *bufptr++;
        scrptr += MAX_WIDTH - pos_width;
        }
hfree(savescreen);
}
```

リスト2：TEST2.Cキャラクタウィンドウプログラムのメイン関数

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main(argc, argv)
int argc;
char *argv[];
{
int x, y, width, raw;
FILE *fp;

if(argc != 6) {
        printf("Usage : test x y width raw filename¥n");
        exit(1);
        }
x = atoi(argv[1]);
y = atoi(argv[2]);
width = atoi(argv[3]);
raw = atoi(argv[4]);
if(!(fp = fopen(argv[5],"r"))) {
        printf("fopen error¥n");
        exit(2);
        }
openwindow(x, y, width, raw, fp);
getch();
closewindow();
}
```

```
glu32 -format lin -stub stub.exe e(test) test1 test2 dos32lib.lib math3r.lib clib3r.lib
```

実際に開発する場合には，コマンドライン上からモジュール名を指定するのではなく，応答ファイルを作っておくことになるでしょう．

-formatオプションは，出力するEXEファイルのフォーマットを指定するオプションで，RationalSystems社が独自に開発したRSI1フォーマット，RSI2フォーマットかOS/2 Ver.2用にMicrosoft社とIBM社で開発されたLINEXEフォーマットのいずれかを選択するものです．Rational Systems社では，32ビットDOSエクステンダ開発にあたって，80386用フォーマットとしてはLINEXEフォーマットを採用しました．その理由は，LINEXEフォーマットが今後の80386用の標準のフォーマットになる可能性が高いこと，LINEXEフォーマットをサポートするユーティリティが開発されればDOS/4Gでも利用できることなどを考慮してのことです．

-stubオプションは，プレローダとでもいうものをアプリケーションに付加するためのオプションです．ここで指定されているstub.exeは，環境変数PATHまたはDOS4GPATHで指定されているディレクトリからDOS4G.EXEを探して，その引数に自分自身を指定して起動をかけるものです．これによって，複数のプログラムを開発するときに，すべてのアプリケーションにDOS4G.EXEを付加することによるディスクの無駄遣いを避け，かつ，アプリケーションは通常のプログラム同様にコマンド名を入力するだけで起動できる環境を提供しています．これで，test.exeというプロテクトモードプログラムができます．

ロードモジュールができるとこれをデバッグすることになります．DOS/4Gには，Instant-D32というフルスクリーンソースレベルデバッガが入っています．これを以下のようにして立ちあげてデバッグをします．

```
d32 test
```

このデバッガの大きさは300KBほどですが，640KB中の300KBではなく，4GB中の300KBですから大きすぎるとはいえないでしょう．デバッガによってバグが見つかれば，再度エデ

メモリ管理用語集
フォールト　命令を実行する前に処理をする例外処理

ィット，コンパイル，リンクを繰り返します．バグがなくなれば，このプログラムにカーネルDOS4G.EXEを組み込んで，単体でMS-DOSから立ち上がる形式にします．DOS4G.EXEがリアルモードとプロテクトモードの橋渡しをするモジュールです．見かけは1つのEXEファイルで，MS-DOSからはDOS4G.EXEしか見えませんが，実はその続きにプロテクトモードプログラムが入っているファイルになるわけです．

DOSエクステンダにおけるメモリ管理

通常，MS-DOSが管理しているメモリは1MB以下のエリアのみですから，1MB以上のエリアについてはDOSエクステンダが独自にメモリを管理し，DOSのメモリ割り付けファンクション48H，49H，4AHを拡張しています．

しかし，1MB以上のメモリを割り付ける機能しかない関数があるわけではなく，MS-DOSのメモリ領域も含めて，メモリ領域全体を管理するようになっています．しかも，Cの標準関数であるmallocやgetmemなどはそれらのファンクションを用いて記述されているので，C言語ユーザーからはその違いがまったく意識されません．

しかし，ときにはプロテクトモードを意識したメモリ割り付けを行う必要があります．たとえば，機種固有のBIOSを呼び出す場合ですが，呼び出すBIOSの入力パラメータにポインタが必要なときに，そのポインタがリアルモードからアクセスできない1MBより上のエリアでは，リアルモードアドレスへの変換ができません．そこで，DOSエクステンダには，メモリ割り付けを行う際に拡張メモリから割り付けるか，MS-DOSのメモリから割り付けるかを指定することができるようになっています．DOS/4Gの場合には，メモリ割り付けを拡張メモリからするか，DOSメモリからするかを指定するファンクションがありますし，386｜DOS Extenderには，DOSメモリ割り付けの専用ファンクションが用意されています．つまり，BIOSを呼び出すときに渡すポインタには，MS-DOSのメモリと指定されたエリアを割り付けておく必要があるのです．しかし，これは特別な場合ですから，このような特殊な場合を除いてはデフォルトの状態で使用して差し支えないでしょう．

プロテクトモード環境のプログラム開発上の問題(一般論)

DOSエクステンダは，プロテクトモードのうち，メモリ領域の拡大機能のみを利用して，そのほかの点では，リアルモードと同じ環境を提供しようとする製品です．できるかぎりリアルモード時の環境を再現するようにしてあるのですが，それでもリアルモードのときには問題にならなかったプログラミング手法が，プロテクトモードの保護機能に引っかかってくることがあります．

I/Oアクセス

in命令，out命令はそのまま使えます．問題はメモリマップドI/Oです．VRAMなどはメモリ空間上にあり，多くのアプリケーションは直接絶対アドレスを指定してそこにアクセスしています．通常プロテクトモードでは，論理アドレスと物理アドレスは一致していませんから，この絶対アドレスの指定ができません．つまり，プロテクトモードで使用するアドレスは，物理的なアドレスとは無関係な値になっています．そこで，DOSエクステンダでは，次の2つの方法でユーザーに物理アドレスを直接操作する手段を提供しています．

【予約されたセグメント】

リアルモードでも，VRAMなどのアドレスは，特定のアドレスに割り当てられています．プロテクトモードでも，VRAMを特定のセグメントまたアドレスに割り付けることによって直接アクセスできるようになっています．

たとえば，DOS/16MやDOS/4Gの場合には，PC-9801のVRAMがセグメントA000H番地にありますから，同じA000H番地にVRAMを割り当てています．このようなセグメントをトランスペアレント(透明な)セグメントと呼びます．つまり，リアルモードとプロテクトモードの区別がつかないセグメントという意味で，透明であるということです．

386｜DOS Extenderの場合には，トランスペアレントにはなっていませんが，セグメント1CHをVRAMに割り当てていますから，A000Hの代わ

メモリ管理用語集
物理ページ　8088CPUでアドレッシング可能な1MB以内の空間の絶対物理空間．論理ページはこのアドレスにマッピングされる．

りに1CHを使えばVRAMに直接読み書きできます．また，DOSエクステンダには，任意の物理アドレスに対応する論理アドレスを作成するためのファンクションもあるので，どんな物理アドレスでもDOSエクステンダ用にプログラムすればアクセスすることができるようなっています．

【リニアアドレス】

DOS/4Gでは，トランスペアレントなセグメントのほかにオフセットアドレスがそのまま物理アドレスと一致するように設定されています．つまり，ポインタにA0000Hを設定すると，物理アドレスのA0000Hを指すように自動的に設定されます．このアクセス方法は，VRAMだけでなく1MB以下の任意のアドレスに対して有効になっています．

DOS/4Gには，2つのVRAMアクセス方法があることになりますが，トランスペアレントなセグメントを使う方法はできるだけ避けたほうがいいでしょう．プロテクトモードでは，セグメントの切り替えにはリニアアドレスをディスクリプタテーブルからディスクリプタテーブルレジスタに読み込み，さらにページテーブルの参照にいきますから，リアルモードのセグメント切り替えより時間がかかります．このようなコストの高い命令はできるだけ避け，32ビットのオフセットアドレスを有効に利用したほうが，32ビットアプリケーションの利点を引き出すことができます．

割り込み

プロテクトモードでは，割り込みは割り込みベクタではなく，IDT（割り込みディスクリプタテーブル）にしたがって処理ルーチンに制御を移します．ですから，何も処理をせずにint21Hを行えば，IDTの21H番目の処理を行うことになります．しかし，MS-DOS上でint21Hが発生する場合，ユーザーが期待するのはMS-DOSのファンクションコールを実行することです．そこで，前述したようにDOSエクステンダのシステムは，int21H割り込みがあった場合には，一度リアルモードに戻り，int21Hを実行しなおします．このとき，MS-DOSのファンクションコールの中には，引き渡されるデータのアドレスを引数として，レジスタに格納しておく必要のあるものがありますが，ユーザーが渡してくるアドレスは論理アドレスになっているので，リアルモードでは無意味なものとなってしまいます．そこで，システムは，int21Hを実行しなおす前に，ファンクションコールに渡されるデータをリアルモードでアクセスできる1MBより下のエリアにコピーし，コピーされたデータを指すリアルモードのアドレスをレジスタに格納してからint21Hを実行します．

このように，DOSファンクションコールのシミュレーションは，カーネルが動的に行っているので，アセンブラルーチンでint21Hを実行しても，ユーザーが意図した通りMS-DOSのファンクションコールを呼び出すことができます．

次に，ユーザーが独自に設定した割り込みを実行する場合を考えます．これは2つの場合に分かれます．1つはそのルーチンをソフトウェア割り込みで実行する場合，もう1つは外部割り込み（ハードウェア割り込み）で実行する場合です．

ソフトウェア割り込みで呼び出す場合，プログラムがプロテクトモードで走っていることはわかっていますから，プロテクトモードのIDTを設定すればいいわけです．そのために，MS-DOSのファンクション25H・35Hは，プロテクトモードの割り込みディスクリプタテーブルを読み書きするようになっています．

ハードウェア割り込みの場合は，割り込みが発生したとき，CPUがプロテクトモードとリアルモードのどちらの状態であっても起こり得るので，どちらのモードで発生してもいいように割り込み処理ルーチンがプロテクトモード用に書かれているのか，リアルモード用に書かれているのかをカーネルに知らせるファンクションを用意しています．たとえば，割り込みルーチンがプロテクトモードで記述してあり，その割り込みがそのファンクションでプロテクトモード用の処理ルーチンが用意されていると指定されていれば，リアルモードで割り込みが発生した場合，プロテクトモードに移行して割り込みルーチンに制御を移すことになります．

また，機種固有のBIOSを呼び出したり，特別なデバイスドライバのファンクションを使う場合のために，リアルモードのアドレスをその処理ルーチンに渡すときに，セグメントレジスタへ任意のデータを格納して（プロテクトモードでは，論理アドレスとして，ディスクリプタテーブルに登録されていないアドレスをセグメントレジスタに代入すると，一般保護例外という実行時エラーを発生して，処理を中断してしまいます），割り込みを発生させるためのファンクションも別に用意されています．

メモリ管理用語集
プロテクトモード　1MB以上のメモリ空間にアクセスでき，メモリ空間の管理・保護，タスク単位でのプログラム管理などの機能をハードウェアで用意する．

コードセグメント変数

割り込み処理ルーチンや割り込み処理ルーチンとデータのやり取りをするルーチンなどでよく行われますが，コードセグメント内に変数領域を確保してそこへ書き込むことがあります．このようなプログラムは，リアルモードでは何の問題もなく動作します．しかし，プロテクトモードでは，コードセグメントへの書き込みは禁止されており，実行すると一般保護例外を起こして処理が中断されてしまいます．このような場合，80286DOSエクステンダでは，コードセグメントと物理アドレスが同じになるデータセグメントを作って，そのデータセグメントを使って書き込みにいくという手法をとらなければなりません．

80386DOSエクステンダの場合にはもう少し簡単です．もともと，コードセグメントのオフセットアドレスとデータセグメントのオフセットアドレスは，同じ物理アドレスを指しているので，コードセグメントを使う代わりに，データセグメントを使って書き込みにいくようにすればいいだけです．

ポインタ計算

リアルモードで64KBを越えるデータをアクセスする場合には，オフセットがオーバーフローしてしまいます．このため，オーバーフローした分をセグメントに繰り込みますが，80386DOSエクステンダではセグメント計算そのものを必要としませんし，80286DOSエクステンダの場合にはセグメントへの繰り上げはあるもののセグメントに加算する値がリアルモードとは異なります．Cのソースレベルでは，この部分はランタイムルーチンに隠れてしまいますから問題はありません．しかし，アセンブラでこのようなことをしているリアルモードルーチンをプロテクトモードに移植する場合には，この部分を書きなおす必要がでてきます．

以上の点は，いずれの場合も特殊なハードウェアにアクセスする場合を除いて，C言語レベル，アセンブリ言語レベルを問わず，MS-DOS環境がプロテクトモードで実現されており，プログラマがプロテクトモードを意識することはほとんどないようになっています．

プロテクトモード環境のプログラム開発上の問題(実際論)

しかし，実際にDOSエクステンダを使ったプログラム開発を行っていくと，前節の問題点がいろいろな形で表れます．とくにこれらの問題に関わってくるのは，市販ライブラリパッケージです．80386DOSエクステンダの場合には，専用の80386Cコンパイラを使うことになりますが，80386Cコンパイラ用のライブラリはあまりありません．80286DOSエクステンダには，DOS用Cコンパイラが使われますが，コードセグメント変数・ポインタ計算の問題があって，リアルモード用のライブラリがそのまま使えるということはまずありません．ソースコードが提供されていれば，それを修正して使用することはそれほどむずかしいものではありませんが，リアルモード用プログラム開発用ライブラリが充実していない点がDOSエクステンダの弱点となっています．ここで，前節で取り上げた問題点の具体的プログラミングについて，PC-9801用DOS/16Mを例として説明します．

I/Oアクセス

I/Oアクセスは，リアルモードプログラムからDOSエクステンダへの移植の中で，比較的簡単な修正になります．場合によっては，オブジェクトまたはライブラリにパッチを当てて修正することができることもあります．

VRAMアクセスの例は，すでにあげたリスト1，2です．これは，VRAMを指すポインタの初期化を処理系に合わせた記述形式に直せばそのままリアルモードでも動きます．

DOSエクステンダのほうで用意していないセグメントでDOSメモリにアクセスする例をリスト3-1，3-2に示します．これは，PC-9801のグラフLIO (GLIO) のベクトルを初期化する部分です．g_init関数の中で，物理アドレスの指定はD16SegAbsolute関数で行われています．第1引数で物理アドレスを指定し，第2引数で割り付けたセグメントの大きさを指定します．0は最大セグメントである65536バイトの指定になります．この関数の戻り値は，第1引数で指定した物理アドレスに割り付けられたセグメントの論理アドレスです．ですから，セグメントの値自体は，物理アドレスとは異なっ

メモリ管理用語集
ページフレーム（Page Frame）　論理ページをマッピングするための物理ページの空間を指す．EMSメモリでは64KBのページフレームが用意される．

リスト3-1：GLIOベクトルの初期化(C-TOOL/98ソースコードより抜粋・修正)

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <dos.h>
#ifdef          LATTICE
#define         BYTE    byte
#else
#include        <malloc.h>
#include        <os2def.h>
#endif
#include        "dos16.h"

typedef unsigned short int word;

#define GLIO_WORK_SIZE   0x1200         /* GLIO用のワークエリアサイズ */
#define PAINT_WORK_SIZE  0xa00          /* PAINT用のワークエリアサイズ */
#define G_WORK_SIZE      (GLIO_WORK_SIZE + PAINT_WORK_SIZE)
#define         DEFAULT (-1)            /* default value              */

#define         high(x)         ((BYTE)(((x) >> 8) & 0xff))

static union {
    BYTE        b[G_WORK_SIZE];
    word        w[(G_WORK_SIZE) / 2];
    struct {
        BYTE    b;
        word    w[(G_WORK_SIZE) / 2 - 1];
        } u;
    } far *glio_work;                   /* グラフＬＩＯワークエリア    */

static word glio_seg;                   /* ワークエリアのセグメント    */
static word glio_off;                   /* ワークエリアのオフセット    */
static BYTE glio_ah;                    /*                             */

#define NEW_GLIO        0xc0            /* NEW-GLIOベクタ              */
#define ID_LEN          8               /* NEW-GLIO識別文字列の長さ    */

#define ERROR           1               /* error code                  */

/**
*
* 関数名                g_lio - グラフＬＩＯルーチンの呼び出し
*
* 書式                  ret = g_lio(vect);
*                       word ret;       グラフＬＩＯの終了条件
*                       int vect;       グラフＬＩＯの内部割込コード
*
**/
static word g_lio(vect)
int     vect;
{
    union REGS regin, regout;
    struct SREGS sreg;

    regin.h.ah = glio_ah;
    regin.x.bx = glio_off;
    sreg.es = sreg.ds = glio_seg;
    int86x(vect, &regin, &regout, &sreg);
    return ((word)regout.x.ax);
}

/**
*
* 関数名        g_init - グラフＬＩＯの初期化
*
* 書式          ret = g_init();
*               int ret;                終了条件
*
**/
int g_init()
{
    int         t, error;
    unsigned    si, di;
    word        doff, dseg;
    word far    *pm_ptr;
    INTVECT     old_vect;
    int         strtgy;
    BYTE        *dummy_iret;


    if(( strtgy = D16MemStrategy( MTransparent )) == -1 ) {
        return 1 ;/* error return */
        }
    if( !(glio_work = D16MemAlloc( G_WORK_SIZE )) ) {
        printf( "Memory allocation error:Not enough DOS memory¥n" );
        return 1;                       /* error return */
        }
    if( !(dummy_iret = D16MemAlloc( 1 )) ) {
        printf( "Memory allocation error:Not enough DOS memory¥n" );
        return 1;                       /* error return */
        }
    if( D16MemStrategy( strtgy ) == -1) {
        return 1;                       /* error return */
        }

    glio_off = FP_OFF(glio_work);
    glio_seg = FP_SEG(glio_work);

    si = 3;
    di = 0xa0;
    dseg = 0xf990;
    if( !( pm_ptr = D16SegAbsolute( (long)dseg << 4, 0 ) )){
        return 1; /*error return */
        }

        /*===========  割込ベクターＡ０Ｈ～ＡＦＨの設定 ===========*/
    for( t=16; t != 0; --t ) {
        doff = *(pm_ptr + si);
        D16rmInstall( di, dseg, doff, NULL );
        si += 2; /* word */
        di++;
        }

    if( !D16SegCancel(pm_ptr) ){
        printf("selecter %p was not canceled.¥n",pm_ptr);
        }

        /*========== 割込ベクターＣ５Ｈの設定 ==========*/
    memset( dummy_iret, 0xcf, 1 );
    doff = FP_OFF(dummy_iret);
    dseg = FP_SEG(dummy_iret);
    D16rmInstall( 0xc5, dseg, doff, &old_vect);

    error = (int)high(g_lio( 0xa0 ));

    g_screen( 0, 0, 0, 1 );
    g_color( 0, 0, 7, 0 );

    return ( error );
}

/**
*
* 関数名        g_term - グラフＬＩＯの終了
*
* 書式          g_term();
*
**/
int g_term() {
    int pal;

    for( pal = 0; pal < 16; ++pal )
        g_color2( pal, 0 );
    g_screen( 0, 2, 0, 1 );
    return (0);
}

/**
*
* 関数名                g_screen - グラフィック画面に対するモード設定
*
* 書式                  ret = g_screen(mod,sw,act,dsp);
*
*                       int ret;        終了条件
*                       int mod;        画面モード
*                       int sw;         画面スイッチ
*                       int act;        アクティブ画面
*                       int dsp;        ディスプレイ画面
*
**/
int g_screen( mod, sw, act, dsp )
int     mod, sw, act, dsp;
{
    int err;

    glio_work->b[0] = (BYTE)mod;
    glio_work->b[1] = (BYTE)sw;
    glio_work->b[2] = (BYTE)act;
    glio_work->b[3] = (BYTE)dsp;

    err = (int)high( g_lio( 0xa1 )) ;
    outp( 0x68, 8 );                    /* graph mode 400 ?    */
    return (err);
}
/**
*
* 関数名        g_color - 背景色等の指定
*
* 書式          ret = g_color(cbg,cbd,cfg,pmd);
*               int ret;                終了条件
*               int cbg;                バックグランドカラー
*               int cbd;                ボーダーカラー
*               int cfg;                フォアグランドカラー
*               int pmd;                パレットモード
*
**/
int g_color(cbg, cbd, cfg, pmd)
int     cbg, cbd, cfg, pmd;
{
    int err;

    glio_work->b[1] = (BYTE)cbg;
    glio_work->b[2] = (BYTE)cbd;
    glio_work->b[3] = (BYTE)cfg;
    glio_work->b[4] = (BYTE)pmd;

    err = (int)high( g_lio( 0xa3 ));
    return (err);
}

/**
* 関数名        g_color2 - パレット番号と表示色コードの対応
*
* 書式          ret = g_color2(pal,cc);
*               int ret;        終了条件
*               int pal;        パレット番号
```


メモリ管理用語集
ポインタ（pointer）　変数の値ではなく変数の格納されるメモリ上のアドレスをもつ。構造体や配列の位置を示すのに使われる。

```
*               int cc;         表示色コード
**/
int g_color2(pal, cc)
int     pal, cc;
{
    glio_work->b[0] = (BYTE)pal;
    glio_work->u.w[0] = cc;
    return ((int)high( g_lio( 0xa4 )));
}
/**
* 関数名        g_line - 直線または矩形を描く
*
* 書式          ret = g_line(x1,y1,x2,y2,pal1,dc,lsw,lstyle,lentile,ptile)
*               int ret;                終了条件
*               int x1,y1;              座標1
*               int x2,y2;              座標2
*               int pal1;               パレット番号1
*               int dc;                 描画コード
*               int lsw;                スイッチ
*               int lstyle;             ラインスタイルまたはパレット番号2
*               int lentile;            タイルパターン長
*               BYTE *ptile;            タイルパターン
**/
int g_line(x1, y1, x2, y2, pal1, dc, lsw, lstyle, lentile, ptile)
int     x1, y1, x2, y2, pal1, dc, lsw, lstyle, lentile;
BYTE    *ptile;
{
    word        doff, dseg;
    BYTE        *tmppmptile;
    int         ret, strtgy;

    if( dc == 2 ) {
        if((strtgy = D16MemStrategy(MTransparent)) == -1){
            return 1; /* error return */
            }
        if(!(tmppmptile = D16MemAlloc(lentile))){
            printf("Memory allocation error:Not enough DOS memory¥n");
            return -1; /* error return */
            }
        if(D16MemStrategy(strtgy) == -1){
            return -1; /* error return */
            }
        memcpy(tmppmptile, ptile, lentile);
        }
    doff = FP_OFF(tmppmptile);
    dseg = FP_SEG(tmppmptile);
    glio_work->w[0] = x1;
    glio_work->w[1] = y1;
    glio_work->w[2] = x2;
    glio_work->w[3] = y2;
    glio_work->b[8] = (BYTE)pal1;
    glio_work->b[9] = (BYTE)dc;
    glio_work->b[10] = (BYTE)lsw;
    glio_work->u.w[5] = lstyle;
    glio_work->b[13] = (BYTE)lentile;
    glio_work->w[7] = doff;
    glio_work->w[8] = dseg;

    ret = ((int)high( g_lio( 0xa7 )));
    if( dc == 2 ) {
        D16MemFree(tmppmptile);
        }
    return ret;
}
```

リスト3-2：GLIOテストプログラム

```
main()
{
g_init();
g_screen(3,0,0,1);
g_line(10,10,200,200,7,2,2,7,6,"¥xaa¥xaa¥xaa¥xaa¥xaa¥xaa");
getch();
g_term();
}
```

た値になります。

リスト3-1，3-2では，この関数を使ってF9900番地にあるGLIOのエントリアドレスを読みにいっています。ここで読んだ値をA0～AFH番のベクトルに設定するために，D16rmInstall関数を使用しています。リアルモードでは，セグメントを0，オフセットをint番号×4番地に直接書きにいくこともできますが，プロテクトモードでは，セグメント0への書き込みはNULLポインタの使用とみなされるので禁止されています。割り込みには，リアルモードの割り込みアドレスが登録される割り込みベクトルと，プロテクトモードの割り込みが登録される割り込み記述子テーブルがあります。それに対応してDOS/16Mには，D16rmInstall関数とD16pmInstall関数があります。DOSのファンクション25H，35Hは，プロテクトモード割り込みのD16pmInstall関数に対応しています。

割り込み

DOSエクステンダのプログラミングの中で，割り込みはもっともむずかしい部分の1つです。これは，プロテクトモードからリアルモードのルーチンとやり取りをしなければならないためですが，プロテクトモードからリアルモードのルーチンを呼び出す方法としては，割り込みという方法はもっとも簡単な方法になります。後述する「GRAPH.SYSの利用方法」では，割り込みを使わずにリアルモードルーチンであるPC-9801のGRAPH.SYSを呼び出す例を説明しますが，ここでは「I/Oアクセス」で設定したベクトルを利用してGLIOを呼び出す例を説明します。

【セグメントレジスタに値を設定する場合】

表1：D16MemStrategy関数の引数

引数	動作
MPreferExt	拡張メモリから先に割り付け，拡張メモリでたりなくなったらDOSメモリから割り付ける。
MPreferLow	DOSメモリから先に割り付け，DOSメモリでたりなくなったら拡張メモリから割り付ける。
MForceExt	拡張メモリのみを使用。DOSメモリは使わない。
MForceLow	DOSメモリのみを使用。拡張メモリは使わない。
MTransparent	DOSメモリのみを使用。割り付けられたメモリのアドレスはトランスペアレントになる。
MTransStack	DOSメモリのみを使用。割り付けられたメモリのアドレスはトランスペアレントになり，STACKセグメント属性となる。

メモリ管理用語集
マッピング（Mapping）　拡張メモリの論理ページを物理ページでアドレッシングするための処理

リスト4：セグメントレジスタに値を設定しないBIOSコール（ビープ）

```
#include <stdio.h>
#include <dos.h>
#include <time.h>
main()
{
union REGS r;
time_t t;

printf("３秒間ブザーを鳴らします。￥n");
t = time(NULL) + 1;
while(t > time(NULL));
r.h.ah = 0x17;
int86(0x18,&r,&r);
t += 3;
while(t > time(NULL));
r.h.ah = 0x18;
int86(0x18,&r,&r);
}

```

g_init関数の最初に使われているD16MemStrategy関数，D16MemAlloc関数は，GLIOとデータのやり取りをするための領域をリアルモードからアクセスできる1MBより下のアドレスに確保しています．D16MemStrategy関数は，その引数によって，この関数が使われたあとのメモリ割り付け関数のメモリの取り方を制御します（表1）．

D16MemStrategy関数は，malloc関数のようなメモリ割り付け関数ですが，malloc関数とは異なり，ライブラリ内に解放されたメモリのバッファをもっていないので，確実にD16MemStrategy関数の指定が有効になります．D16MemStrategy関数では，MForceLowではなくMTransparentを指定していますが，これは，GLIOに渡すときにDSレジスタに代入する値は論理アドレスではなく，リアルモードで使える物理アドレスでなければならず，しかも，g_lio関数の中で使っているint86x関数では，DSレジスタに値が格納されるのがプロテクトモードであり，論理アドレスとしても意味をもっていなければならないためです．

MForceLowでメモリを確保する場合には，D16MemAlloc関数が返してきた論理アドレスをD16RealPtr関数でリアルモード用のアドレスに変換して，GLIOを呼ぶときもD16rmInterrupt関数を使ってDSレジスタへの値の格納で実行時エラーが起こらないようにしなければなりません．また，ここで注意しなければならないのは，GLIOではESレジスタの値を使っていないので，リアルモードのときはint86x関数の構造体SREGSのメンバesは初期化しなくてもよかったのですが，int86x関数はメンバesの値をesレジスタに読み込みますから，プロテクトモードではそこに不定な値が入っていると実行時エラーが起こります．このため，必ず何かの値で初期化しなければなりません．

【セグメントレジスタに値を設定しない場合】

リスト4にint18Hファンクション17H，18Hを使ってブザーを鳴らす例をあげます．セグメントレジスタに値を設定しないので，int86関数が使えます．セグメントレジスタでパラメータを渡さない場合には，この例の用にリアルモードと同様にファンクションを呼び出せます．

コードセグメント変数

コードセグメントにあるデータを読み出すことはできますが，そこに書き込むことはできません．cs:プレフィックスを指定したり，コードセグメントをdsレジスタやesレジスタに入れてデータを書き込むことはできませんが，コードセグメントと同じ物理アドレスにデータセグメントを割り当てて書き込むことはできます．

リスト5-1，5-2にその例として，タイマ割り込みの処理ルーチンと，メインルーチンとのデータをやり取りするエリアをコードセグメントに取ったプログラムをあげます．やっていることはつまらないことで，タイマ割り込みで共通領域のデータに1ずつ加えていき，メインルーチンで共通領域の値を画面に表示しつつ，キー入力があったら共通領域を0にするというものです．

リスト5-1のアセンブラルーチンのほうは，アセンブル時に-dREALを指定するとリアルモード用になるようにして比較できるように書いてあります．リアルモードでは，“inc _counter”のようにコードセグメントのデータに書き込みできますが，プロテクトモードではポインタ変数dsaliasに格納されているポインタを使います．これは，物理アドレスとしてコードセグメントと同じ領域を指すデータセグメントのポインタです．このような同じ領域を別の論理アドレスで指すことをエイリアスといいます．データセグメントですから，書き込みは自由にできます．この変数dsaliasは，main関数で初期化されています．D16SegDSAlias関数は，引数のポインタが指すコード領

メモリ管理用語集
メモリウィンドウ（Memory Window）　ページフレームの別の呼び方．

リスト5-1：コードセグメント変数の回避方法(セグメントエイリアスの使用)

```
ifdef   REAL
        .286p
endif
_TEXT   segment para    public  'CODE'
        assume  cs:_TEXT
        public  _counter
_counter        dw      0
ifndef  REAL
        public  _dsalias
_dsalias        dd      0
endif
        public  _t_handler
_t_handler      proc    far
ifndef  REAL
        push    ds
        push    bx
endif
        push    dx
        push    ax
ifdef   REAL
        inc     _counter
else
        mov     ax,_counter
        inc     ax
        lds     bx,_dsalias
        mov     ds:[bx],ax
endif
        mov     al,30h
        mov     dx,3fdfh
        out     dx,al
        jmp     short $+2
        jmp     short $+2
        jmp     short $+2
        mov     al,0ffh
        mov     dx,71h
        out     dx,al
        jmp     short $+2
        jmp     short $+2
        jmp     short $+2
        out     dx,al
        jmp     short $+2
        jmp     short $+2
        jmp     short $+2
        mov     al,20h
        out     0,al
        pop     ax
        pop     dx
ifndef  REAL
        pop     bx
        pop     ds
endif
        iret
_t_handler      endp
_TEXT   ends
        end
```

リスト5-2：コードセグメント変数の回避方法(セグメントエイリアスの使用)

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "dos16.h"
extern unsigned short int counter;
extern unsigned char far *dsalias;
extern  t_handler();
main()
{
union   {       unsigned short int r16[2];
                unsigned char far *ptr;
        } xptr;
INTVECT rtvec;
unsigned short int imr;
int c;
char far *tmp;
tmp = D16SegDSAlias((void (far *) ())&counter);
*(unsigned char far * *)(tmp + 2) = tmp;
D16rmGetVector(0x08,&rtvec); /* オリジナルのベクトルの保存 */
xptr.ptr = (unsigned char far *)t_handler;
D16pmInstall(0x08,xptr.r16[1],xptr.r16[0],NULL);
                /* プロテクト・モードの割込みテーブルの設定 */
D16Passup(0x08); /* リアル・モードで割込みが起こった場合、*/
                /* プロテクト・モードの処理ルーチンを使うことを指定 */
outp(0x3fdf,0x30); /* インターバル・タイマー8253の設定 */
outp(0x71,0xff);
outp(0x71,0xff);
outp(0x02,(imr = inp(0x02)) & 0xfffe); /* タイマー割込み許可 */
for(;;){
        if(kbhit()){
                c = getch();
                if(c == 'Z') break;
                *dsalias = 0;
                }
        printf("%d¥n",*dsalias);
        }
outp(0x02,imr); /* タイマー割込み禁止 */
D16rmInstall(0x08,rtvec.sel,rtvec.off,NULL);
}
```

リスト6-1：アセンブラルーチンによるポインタの計算

```
ifdef LATTICE

extrn   _SINC:word
else
extrn   __AHINCR:near
ensdif
ifdef LATTICE
DGROUP  group   _DATA
_DATA   segment word    public  'DATA'
        assume ds:DGROUP
_DATA   ends
endif
_TEXT   segment para    public  'CODE'
        assume  cs:_TEXT
        public  _inc_far_pointer
_inc_far_pointer        proc    far
        push    bp
        mov     bp,sp
ifdef LATTICE
        mov     bx,[bp + 6]
        mov     ax,[bp + 8]
        add     bx, 1
        jnc     _inc_far_pointer_end
        add     ax,_SINC
else
        mov     ax,[bp + 6]
        mov     dx,[bp + 8]
        add     ax, 1
        jnc     _inc_far_pointer_end
        add     dx,offset __AHINCR
endif
_inc_far_pointer_end:
        pop     bp
        ret
_inc_far_pointer        endp
_TEXT   ends
        end
```

リスト6-2：アセンブラルーチンによるポインタの計算

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include "dos16.h"

void far * inc_far_pointer(void far *);

main()
{
unsigned char huge *ptr, huge *ptr2;
if(!(ptr = D16HugeAlloc(0x20000))){
        printf("D16HugeAlloc error¥n");
        exit(1);
        }
ptr2 = ptr + 0x10002;
for(FP_OFF(ptr) = 0xfffe; ptr < ptr2; ptr = inc_far_pointer(ptr)){
        *ptr = 'A';
        printf("%lp¥n",ptr);
        }
}
```

域と同じ領域を指すデータセグメントへのポインタを返します．変数dsalias自身もコード領域にありますから，D16SegDSAlias関数が返してきたポインタを使って初期化しています．このとき，変数counterと変数dsaliasは並んでいるので，counterへのエイリアスポインタを使ってdsaliasへの代入を行っていますが，もう一度D16DSAlias関数を使って別のエイリアスポインタで代入してもかまいません．

メモリ管理用語集
メモリ保護　マルチタスクにおいて，各タスクのアプリケーションがほかのアプリケーションのコードやデータを破壊するのを防ぐこと．

この逆に，データセグメントでのコードの実行もリアルモードでは自由にできますが，プロテクトモードでは同様にD16CSAlias関数でコードセグメントをエイリアスして実行することができます．

ポインタ計算

Cのソースコードならば，コンパイル，リンクをしなおすだけです．アセンブラで書かれた部分で，どうしてもfarポインタを計算しなければならない場合には，リスト6-1のようにして行います．この方法は，OS/2でも使用できます．この例は，ポインタを1増加して返す関数ですが，オフセットがオーバーフローしたときにセグメントの値がいくら増加するかは，Lattice Cでは外部変数_SINCの値で，MS-Cでは外部変数__AHINCRのオフセット値で表されます．この値をオフセットのオーバーフロー時にセグメントに加えるという処理を行っています．

リアルモード時と異なるのはセグメントに加える値だけですから，手間を除けば比較的対応の簡単なものです．

GRAPH.SYSの利用方法

PC-9801用MS-DOSには，グラフィック描画用のデバイスドライバ

リスト7：GRAPH.SYSの利用方法

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "dos16.h"
union GrpDataTag {
        unsigned int work[1024];
        struct PsetTag {
                unsigned char far *Reserved;
                unsigned int Rop;
                unsigned char ActMode;
                unsigned int X;
                unsigned int Y;
                unsigned int Clr;
                } Pset;
        struct DspSw {
                unsigned long Reserved;
                unsigned char Switch;
                } DspSw;
        struct RectTag {
                unsigned char far *Reserved;    /* 0*/
                unsigned int Rop;               /* 4*/
                unsigned int X1;                /* 6*/
                unsigned int Y1;                /* 8*/
                unsigned int X2;                /* a*/
                unsigned int Y2;                /* c*/
                unsigned char DrawF;            /* e*/
                unsigned char dum1;             /* f*/
                unsigned long Pal;              /*10*/
                unsigned int width;             /*14*/
                unsigned char PatLength;        /*16*/
                unsigned char dum2;             /*17*/
                unsigned int pat;               /*18*/
                unsigned int pat2;              /*1a*/
                unsigned char FillF;            /*1c*/
                unsigned char dum3;             /*1d*/
                unsigned long FPal;             /*1e*/
                unsigned int TileLength;        /*22*/
                unsigned long Tile;             /*24*/
                } Rect;
        } far Grp_Data;
static int dummy_for_ds;

main()
{
struct FuncEntryTag {
        int (pascal far *func[41])(void far *);
        } far * GL;
D16REGS r;
char far * tmpptr;
int ret;
FILE *fp;
/* Variable for DOS/16M */
unsigned r_cs, r_ss, r_ds, r_es;
int mst;
D16INFO *_d16inforp;
union GrpDataTag far * Grp_Data_rp;

/* initialization for DOS/16M */
mst = D16MemStrategy(MForceLow);
        /* Reallocate far data segment to DOS memory */
if(!D16SegRealloc(&Grp_Data)){
        printf("Error D16SegRealloc¥n");
        exit(1);
        }
        /* Reallocate stack segment to DOS memory */
if(!D16SegRealloc(&mst)){
        printf("Error D16SegRealloc¥n");
        exit(1);
        }
        /* Reallocate code segment to DOS memory */
if(!D16SegRealloc(main)){
        printf("Error D16SegRealloc¥n");
        exit(1);
        }
D16MemStrategy(mst);
_d16inforp = D16RealPtr(&_d16info);
Grp_Data_rp = D16RealPtr(&Grp_Data);
tmpptr = (char far *)&mst;
r_ss = FP_SEG(tmpptr);          /* Protected mode stack segment */
tmpptr = (char far *)&dummy_for_ds;
r_ds = FP_SEG(tmpptr);  /* Protected mode data segment */
tmpptr = (char far *)&Grp_Data;
r_es = FP_SEG(tmpptr);          /* Protected mode extra data segment */

/* Check for GRAPH.SYS */
if(!(fp = fopen("GRAPH  $","r"))){
        printf("No graphic driver¥nAbort ? ");
        ret = getch();
        printf("¥n");
        if(toupper(ret) == 'Y')
                exit(1);
        }
else fclose(fp);

/* Get Entry point */
tmpptr = (char far *)D16RealPtr(&GL);
r.bx = FP_OFF(tmpptr);
r.ds = FP_SEG(tmpptr);
r.ax = 0;
D16rmInterrupt(0xcd,&r,&r);

/* Initialize */
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[0])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[0] ret=%d¥n",ret);
#endif

/* Set display mode */
Grp_Data.Rect.Reserved=(char far *)NULL;
Grp_Data.work[2]=0x101;
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[3])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[3] ret=%d¥n",ret);
#endif

/* Set Draw plane */
Grp_Data.Rect.Reserved=(char far *)NULL;
Grp_Data.work[2]=7;
Grp_Data.work[3]=0;
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[4])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
```


メモリ管理用語集
メモリマネージャ（Memory Manager）　拡張メモリやプロテクトメモリなどを複数のアプリケーションから利用する場合に競合しないようメモリを管理するプログラム．

```
_intflag(1);
#ifdef DEBUG
printf("GL->func[4] ret=%d¥n",ret);
#endif

/* Set Display plane */
Grp_Data.work[2]=7;
Grp_Data.work[3]=0;
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[5])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[5] ret=%d¥n",ret);
#endif

/* Set view port area */
Grp_Data.Rect.Reserved=(char far *)NULL;
Grp_Data.work[2]=0;
Grp_Data.work[3]=0;
Grp_Data.work[4]=639;
Grp_Data.work[5]=399;
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[7])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[7] ret=%d¥n",ret);
#endif

/* Set display switch */
Grp_Data.Pset.Rop=1;
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[11])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[11] ret=%d¥n",ret);
#endif

/* Draw rectangle */
Grp_Data.Rect.Reserved=(char far *)0;
Grp_Data.Rect.Rop=0;
Grp_Data.Rect.X1=100;
Grp_Data.Rect.Y1=100;
Grp_Data.Rect.X2=199;
Grp_Data.Rect.Y2=199;
Grp_Data.Rect.DrawF=0x10;
Grp_Data.Rect.Pal=7;
Grp_Data.Rect.width=0xf;
/*Grp_Data.Rect.PatLength;*/
/*Grp_Data.Rect.pat;*/
/*Grp_Data.Rect.pat2;*/
Grp_Data.Rect.FillF=1;
Grp_Data.Rect.FPal=7;
/*Grp_Data.Rect.TileLength;*/
/*Grp_Data.Rect.Tile;*/
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[18])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[18] ret=%d¥n",ret);
#endif

getch();

/* terminate graphics */
if(!(r_cs = D16ToReal())){
        printf("Cannot go to real mode¥n");
        exit(1);
        }
ret=(*(int (pascal far *)())GL->func[1])((void far *)Grp_Data_rp);
/*D16ToProtected(r_cs, r_ss, r_ds, r_es);*/
(*_d16inforp->D16ToPMP)(r_cs, r_ss, r_ds, r_es);
_intflag(1);
#ifdef DEBUG
printf("GL->func[1] ret=%d¥n",ret);
#endif
}
```

GRAPH.SYSがあります．このGRAPH.SYSは，intCDHを使ってファンクション呼び出し用のアドレステーブルを得ます．描画したい図形に対応するアドレスをこのテーブルから選んでpascal形式で呼び出すことによって，グラフィック表示ができます．

DOS/16Mをはじめ，たいていのDOSエクステンダでは，割り込みが起こるとリアルモードになるのがデフォルトの状態で，モード切り替えに割り込みを利用すると便利になっています．しかし，GRAPH.SYSではcall命令を使うので，呼び出すためにはモードを切り替えてから呼び出さなければなりません．まず，リアルモードに戻るためには，戻るときに実行されているコードセグメント，データセグメント，スタックセグメントが1MBより下のアドレスになければなりません．そのため，D16MemStrategy関数をMForceLowに指定し，D16SegRealloc関数で各セグメントを1MBより下のアドレスに割り当てなおします．MTransparentではなくMForceLowを指定しているのは，MTransparentではコードセグメントなどリンカによって静的に論理アドレスがリンクされているものは，D16SegRealloc関数で再割り付けできないためです．プロテクトモードからリアルモードに移行する関数は，D16ToReal関数です．リアルモードに移ったあとには，以下の制約があります．

①farコールしてはならない．

②ds, esレジスタには論理アドレスが入っているので，対応するリアルモードアドレスに変換しなければならない．ローカル変数（ssセグメントベース）はアクセス可能である．

変数GLには，リアルモードアドレスに変換した関数アドレステーブルへのポインタをあらかじめ代入してありますので，このGLを使ってGRAPH.SYSを呼び出します．GRAPH.SYSの処理が終了したら，_d16info.D16ToPMP関数を使ってプロテクトモードに戻ります．_d16 inforpには，_d16infoへのリアルモードポインタが入っています．このときに各セグメントレジスタの論理アドレスの値を引数にしますので，リアルモードに移る前にこれらの値をローカル変数に待避しておきます．

以上のように若干の制約はありますが，プロテクトモード／リアルモードの切り替えもできますから，かなり柔軟性のあるプログラミングができます．

メモリ管理用語集

メモリモデル　8086CPUのメモリ空間は64KB単位のセグメントで表現される．そのためデータ領域やコード領域の64KBを境にして複数のメモリモデルが存在する．

バーチャルメモリマネージャVMMの動作原理

実際にパソコンに載っているメモリよりも大きなプログラムを動かしたいという場合に，ディスク領域にメモリのイメージを退避しておいて実行する仮想記憶という方法があります．DOSエクステンダに仮想記憶をサポートするオプション製品を使うとこの仮想記憶が使えるようになります．

80386のページテーブルには，そのページが現在メモリ上にあるのか，ディスク上にスワップアウトされているかを示すフラグがあります．このフラグがディスクにスワップアウトされている状態を示しているページに対して，プログラムがアクセスにいくと(そのセグメントのデータをリード・ライトするか，ルーチンをコールあるいはジャンプすると)，80386はトラップするようになっています．

386DOSエクステンダは，この機能を使ってトラップが起きたときにディスク上からそのページをロードしてきます．もし，メモリが不足していれば，使用されていないページをディスクに書き出してメモリを確保します．また，メモリ上にあるセグメントで書き換えが行われていないページの場合（おもにコードセグメントに属するページですが，前述したようにコードセグメントであっても，物理的には同じアドレスを別の論理アドレスとしてデータセグメントに割り付けて書き換えるというデータエイリアスは可能ですから，コードセグメントであっても書き出すことがあります)には,すでにディスク上に同じイメージがあるわけですから，書き出す必要がないわけです．それを省略するという判断も，80386DOSエクステンダで行っています．

表2：おもなDOSエクステンダの対応機種

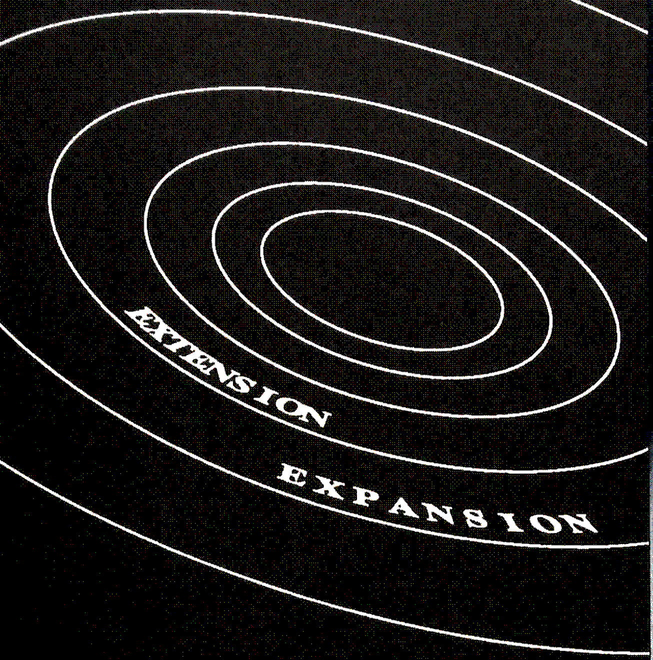
DOS/4G	NEC PC-9801シリーズ
	東芝J-3100シリーズ（デバッガは，英語DOSのみ）
	IBM PC/AT・PS/2互換機
	各社AX仕様機（デバッガは，英語DOSのみ）
386 | DOS Extender	NEC PC-9801シリーズ
	富士通FMTowns
	富士通FMRシリーズ
	IBM PC/AT・PS/2互換機
DOS/16M	NEC PC-9801シリーズ
	富士通FMRシリーズ
	東芝J-3100シリーズ（デバッガは，英語DOSのみ）
	IBM PC/AT・PS/2互換機
	各社AX仕様機（デバッガは，英語DOSのみ）
	日立B16／B32シリーズ
	沖電気if800／RX／EXシリーズ

80286DOSエクステンダに対しても，同様な仮想記憶をサポートするオプション製品というものがありますが，80286の場合にはページングの機能がないので，セグメント単位でディスク上への待避・復帰を行います．仮想記憶は，ページング機構があったほうが効率よく行えます．なぜなら，あるセグメントにある1バイトのデータを読みにいくだけでも，セグメント単位では最大64KBをディスクから読み込んでこなければなりませんが，ページング機構があれば4KBですむからです．

このように，ディスクスペースを仮想的にメモリと見なしてメモリの不足を補うという仮想記憶を，DOSエクステンダにオプション製品を追加して使用することで実現しています．これによって，メモリが十分にあれば，すべてオンメモリで高速に動作し，メモリは不十分でも，ハードディスクがついていれば，大きなメモリを必要とする高機能なプログラムが動作するというように，環境に対してのフレキシビリティがDOSエクステンダに加わります．

以上がDOSエクステンダの概要です．プログラマにプロテクトモードを意識させずに，メモリの大きなリアルモードのようにプログラミングできる環境を提供するために，このようなさまざまな工夫がされているわけです．

メモリ管理用語集
UMB（Upper Memory Block）　640KBから1MBのROM空間．ここにRAMをマッピングし，TSRやデバイスドライバを移動し，メインメモリを広くすることが可能．



特集

286/386メモリ活用プログラミング

プログラミングはどう変わるのか

フリーエリアが広がった MS-DOS 5.0

インタビュー●
マイクロソフト株式会社 技術支援部　中嶋雄一

MS-DOS 5.0が米国マイクロソフト社からリリースされたというニュースは，すでに読者の耳にも届いていることと思う．国内では，日本アイ・ビー・エムやエプソンなど，一部のメーカーをのぞいて，MS-DOS Ver.4.0への移行は行われなかったが，今回のバージョンアップには，日本電気をはじめ国内の各メーカーとも対応を表明している．新しいDOSではどのような点が変わったのか，プログラミングに与える影響はどの程度あるのか，MS-Windowsと新しいMS-DOSの関係はどうなるのか，また各パソコンメーカーによるチューンナップで，どの程度の機能の違いが出てくるのだろうか．マイクロソフト株式会社技術支援部の中嶋雄一氏にお話をうかがった．

HMAとUMBの利用でフリーエリアが広がった

――まず，MS-DOS5.0ではどのような機能が追加されたのでしょうか

MS-DOS 5.0（以下DOS5）では，メモリが広くなったといわれています．フリーメモリを広くするために，いくつかのことをやっているわけですが，まず，はじめて前のバージョンよりもDOS自身のプログラムサイズが小さくなったという特徴があります．具体的には，4.01よりもカーネルのサイズがかなり小さくなっています．

カーネルが小さくなったということは，640KBのなかでプログラムから使えるメモリが広がることになります．さらに，XMS規格でいうHMA*1にDOSのカーネル自体を移して，640KBのなかに占めるDOSのエリアを取り除くことによって，フリーエリアを広げています．もうひとつは，UMB*2の利用があげられます．これもXMSの規格のひとつで，具体的には640KBから1MBまでの空いているところを，RAMとして割り当てることによって使えるようにするというものです．これを使うことによって，アプリケーションから使えるメモリが増えました．

中嶋雄一氏

ただし，日本語版の場合には，DOSをマイクロソフトから直接エンドユーザーには販売しておりません．ハードウェアメーカー様が自社のハードウェアにアダプテーションした形で出荷されますので，製品になったときにメモリサイズがどうなるかという，具体的なことはこちらではわからないのです．4.01を出しているところでは，50，60KBから100KB程度メモリが広く使えるようになると思います．日本電気さんでは，DOS4.01は出荷されておりませんので，4.01との比較というのはできません．日本電気さんの3.3というバージョン―これは私どもでは3.21というバージョンになりますけれども―これと比べますと，カーネルサイズは増えてしまっていますが，その分を差し引いてもHMAやUMBの利用によって，30～40KBは確実にフリーエリアが増えるのではないかと思います．

＊1　High Memory Area の略．1MB 超の64KB のメモリ領域を指す（図1参照）

＊2　Upper Memory Block の略．

――メモリ関連以外では，どのような機能がつけ加えられたのでしょうか．

Windowsライクなビジュアルを持つシェルがつけ加えられています．このシェルは，グラフィックモードとテキストモードの切り替えができます．グラフィックモードでは，Windowsに近い画面イメージで，ドライブアイコンなども表示されます．ほかにも，タスクスイッチング機能，それに付随したかな漢字変換の制御の追加などがあります．あとは，各コマンドにつきまして，簡単なオンラインヘルプが出るようになっております．たとえば

```
A>DIR /?
```

としますと，DIRのオプションスイッチの一覧と，DIRそのものが何をするコマンドかという説明が表示されます．DIRコマンド自体も拡張されております．たとえば，日付順に並び替えて表示する，HIDDENファイルだけ見る，システムファイルだけ見る，そのディスクの中のサブディレクトリも検索して表示するなど，新しい機能が追加されています．ほかに，削除したファイルの復活（UNDELETE）や，間違ってフォーマットしてしまった場合のリカバリ（UNFORMAT）もできるようになっています．もちろん，消してしまったすぐ後，フォーマットしてしまったすぐ後に限るという制限はあります．

SETVERという機能も追加になりました．たとえば，バージョンが変わっても問題なく動くようなアプリケーションで，単にバージョンチェックだけを行っているために動作しないものに対して，DOSのほうから返すバージョン番号を合わせてやる仕組みです．これを使えば，バージョンチェックでDOS4でしか動かないようになっているアプリケーションが，DOS5の上で動くようになるというわけです．

あとひとことつけ加えさせていただければ，US版のマニュアルの冒頭にはDOS5の位置づけとして，Windows 3.0を使うためのベストな環境であるという表現があります．

メモリ管理用語集
UMBローダ（UMB Loader）　UMB領域にTSRやデバイスドライバを組み込むためのプログラム．

DOS5はWindows 3.0を使うために最適の環境

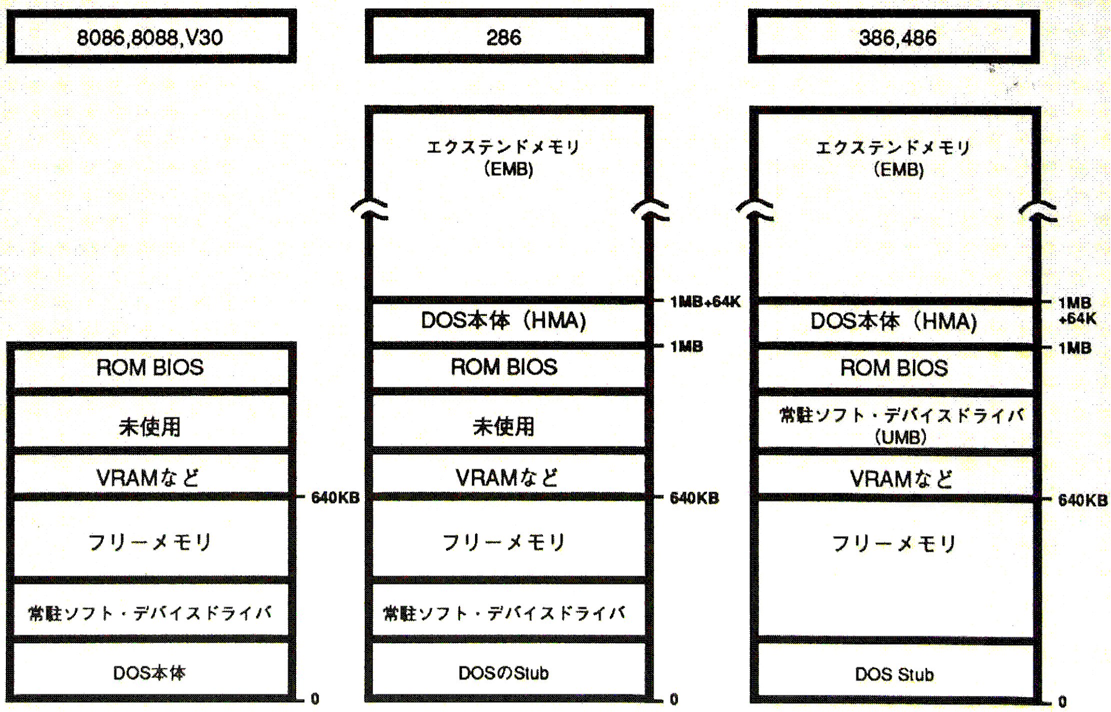
――Windowsにベストというのはどういう点についてでしょう．メモリが広いというところからきているのでしょうか．

Windowsのリアルモードでは，メモリ容量によってアプリケーションが動かない場合もあるかと思いますが，その場合はフリーエリアが大きく関係しています．あればあるほど快適に動きます．そういった意味ではまずDOS5がベストです．WindowsでDOSボックスを利用する場合は，リアルモード，スタンダードモードとも，Windowsを起動する前とまったく同じ環境になります．当然，もとのフリーエリアが広いほどDOSアプリケーションには有利になります．これがほかのバージョンのDOSですと，メモリが圧迫された状態でDOSボックスがたち上がることになります．エンハンストモードの場合では，DOSボックスは新しくDOSが起動されるというイメージになりますが，リアルモード，スタンダードモードと同様の効果が得られます．

エンハンストモードについては，EMM386[*3]とHIMEM.SYS[*4]はWindows 3.0よりもDOS5のほうが内部的には新しいバージョンになっています．DOS4.01にもSMARTDRV.SYS[*5]，HIMEM.SYS，RAMDRV.SYSとEMM386がついていますが，DOS4のSMARTDRVはWindowsの上で独自に動いてしまうのです．DOS5のSMARTDRVはWindowsを起動すると，自分のアロケートしていた領域をWindowsにいったん預けます．ですから，メモリを4MBしか積んでいないマシンで，SMARTDRVが2MBを取っているとすると，DOS4では残り2MBのメモリでWindowsが動く形になりますが，DOS5のSMARTDRVですと，SMARTDRVが取っていた2MBも含めてWindowsが使えるようになります．EMM386も同じです．自分が

図1

MS-DOS Ver.5のメモリマップ



メモリ管理用語集

リアルモード　8086と同様にセグメントとオフセットでメモリにアクセスする．最大1MBのメモリ空間しかアクセスできないのも8086と同じ．

使っていたメモリを，情報としてWindowsに渡すことによって，エンハンストモードのWindowsはEMM386がハンドリングしていたメモリをすべて自分が肩代わりするようになります．

＊3　EMM386.SYS／EMM386.EXE　エクステンドメモリにEMSメモリをシミュレートし，386/486CPUにUMBへのアクセスを提供するドライバ．Windows3.0では拡張子は.SYSであったがDOS5.0では.EXEに変更．単独のコマンドとして実行すればメモリの使用状況を表示する．

＊4　HIMEM.SYS　エクステンドメモリとHMAの使用を管理するメモリマネージャ．

＊5　SMARTDRV.SYS　エクステンドメモリまたはEMSメモリにディスクキャッシュを作成するドライバ．

プログラマから見たDOS5の変更点 4.01からの追加

――では，プログラマから見た場合はどのような点が変わっているのでしょうか

フリーエリアが広がったことによるメリットは，そのままプログラマの方にもあてはまると思います．実は，UMBやHMAの利用については，EMSも含めて，サードパーティさんからはすでに製品化，出荷されております．たとえば，MEMORY-PRO[＊6]ですね，あとメルウェアや，I・Oデータさんの製品などいろいろあります．

DOS5は，それらと比べて決定的な違いがひとつあります．これらの製品では，あくまでもユーティリティだけでUMBの使用が実現されているのです．つまり，アプリケーションがDOSのメモリ領域をアロケートするファンクションを発行すると，640KBの中からしかアロケートできなかったわけです．それがDOS5の場合では，UMBを含めてアロケートするという指定がアプリケーションプログラム側から発行できるのです．これまでは640KBの中で，下からとっていくか，上からとっていくかという選択しかなかったわけですが，DOS5ではUMBを優先してとっていくかどうかという指定ができるようになっています．

たとえば，MEMORY-PROではLU.COM, LU.DRVなどのTSRやデバイスドライバが必要で，それを使わなければ，UMBを利用することができなかったわけですが，DOS5の場合ですとUMBが存在すれば，アプリケーションが自分でメモリを取りにいくことができる，この点がプログラマの方から見た一番大きなメモリ管理上の違いだと思います．DOS5が動いている環境では，必ずしもUMBがあるというわけではないと思いますが，ユーザーがUMBを使う・使わないという指定ができますので，プログラマにとってかなり便利な機能だと思います．

＊6　MEMORY-PRO386．仮想EMS，仮想UMBなどの機能を実現したメガソフト社製のメモリマネージャ．

――MS-DOS5.0で新たにつけ加えられたファンクションや内容が変更になったファンクションがありますか

4.01との比較では，ファンクションレベルで変わったのは2つ3つ程度です．メモリアロケートのストラテジをどうするかということと，バージョンチェックで，ROMバージョンであるかどうかという違いを拾ってこられるようになったこと，それからDOSがHMAにあるかどうかということを知ることもできるようになりました．

ただし，米国ではアプリケーションがほとんどDOS4に対応していますので，DOS5への対応の影響はほとんどないと思いますけれど，日本では事実上の標準が日本電気のMS-DOS3.3です．つまり，私どもでいう3.21に比べると変更はたくさん入っております．

DOS4以降は，ハードディスクの1パーティションが2GBまで拡張されています．PC-9800シリーズの場合は，すでに日本電気さんのMS-DOS Ver.3.3で1パーティション128MBまで拡張されていますが，DOS5の上でどうするのかというのは日本電気さんの判断になるでしょう．DOS5のメリットを活かすという点からすると，日本電気さんはいままでの方式とDOS5の方式を共存，もしくは両立させる方向に進まれるのではないかと思います．もちろん実際の製品が出てみないと確かなことはいえませんが，そういった点でもプログラミングへの影響はあるかと思います．

――これまで，位置が固定されていたDOSカーネルがHMAに置かれたり，デバイスドライバやTSRがUMBに置かれたりする可能性があるということで，プログラマがこれまで使っていた手法が問題になるということも考えられるでしょうか．

そういう可能性もあるかと思いますけれど，デバイスドライバのデバイスリンクのようなものは，リンクを使ってたどっていただく限り影響は出ないようになっています．リンクポインタをたどっていけば，どこに入っているかという情報が返ってきますから，上にあっても下にあっても関係ありませ

メモリ管理用語集
リニアアドレス　セグメントユニットが出力する32ビットの番地

ん．DOSのStub（ヘッダファイルの集まりと考えていい）は，いままでどおりの場所に格納してありますので，よほどおかしな作り方をしていない限り，プログラムはいままで通り動くように工夫されております．DOS自体が上にいったといっても，640KBの中に15KB～20KB弱はStubの集まりなどの情報が残されています．そこを使う限りは，問題は出ないと考えております．

日本語版タスクスイッチャにはかな漢字変換の制御ファンクションを追加

――新しくつけ加わった機能として，タスクスイッチャがありますが，これはいまのところ，シェルをたち上げないと利用できないようになっていますが．

そうです．タスクスイッチはDOSシェルの上で起動する形になります．マルチタスクにはなっておりません．いままで動いていたタスクをフリーにして，ディスクにスワップアウトして，新しいものを起動する，もしくは切り替えるという形です．これは，Windows3.0のリアルモードとスタンダードモードのタスクスイッチそのものです．

――プログラマーズリファレンスを見ますと，一応タスクスイッチングのファンクションも公開されていますね．

ええ，ただしそれはDOSシェルが立ち上がっていないとアクティブにはなりません．今回の米国バージョンですと，タスクスイッチというのはタスク上で常駐プログラムを入れても，次のタスクに移ったときには利用できなくなってしまいます．日本語版の場合は，かな漢字変換がタスク上でどうなるのかという問題がありますので，これについては，かな漢字変換制御用のファンクションを新たに追加しまして，DOSシェルを起動する前に組み込んでおいたかな漢字変換を，各タスク間で問題なく使えるような仕組みをつけ加えております．あるタスクで，かな漢字変換中にタスクを切り替えても，動作は保証できるようになっています．

また，メモリを圧迫するのを覚悟のうえで，DOSシェルをたち上げる前に，複数のかな漢字変換を登録しておいて，各タスク間で異なったかな漢字変換を使えるようにする仕組みもつけ加えています．こうしておかないと「一太郎」を使いながら，VJEを使ったタスクに切り替えるということができなくなってしまいますから．

――DOS5の機能を生かしたアプリケーションを作成するポイントがありましたら教えてください．

やはり，UMBの利用というのが大きいと思います．UMBはDOS5についてくる標準のものでは，CPUが386でないと利用できないということになっています．XMSのマネージャは，HIMEM.SYSですが，実際にメモリをアロケートするのはEMM386がやっているからです．ただし，CPUが286やV30，86の場合でも，サードパーティさんでドライバを用意していただければDOSのほうでは利用可能になります．

私どもの認識では，国内のパソコンでは，UMBとして取れる領域は少ないのではないかと思います．EMSとの併用の場合ですと，多分64KBとれればいいほうだと思います．ネットワークドライバなどを組み込んでいる場合には，さらに少なくなってしまいますから，32KBくらいしか残らないでしょう．それでも，メモリアロケートする場合に，UMBを優先してとるようにすればいいと思います．UMBで確保できる領域がなければ，下のほうから自動的にとってきてくれますので，エラーのリカバリで頭を悩ますということはないでしょう．

386CPUの場合には，EMM386が管理していますから，EMSもサポートされている可能性が高いでしょう．というのは，EMM386ではUMBだけ使ってEMSは使わないという指定もできるからです．そのため，必ずしもEMSとUMB両方使えるとは限らないのですが，可能性としては両方使える場合が多いと思います．EMSを使いながらUMBを使うという方法でいけば，かなりメモリに関しては楽になると思います．

メモリ管理用語集
ローカルアドレス　プロテクトモードでは，ほかのタスクとのメモリのバッティングやOSの保護を目的としてグローバルアドレスを参照することはできない．

パソコン博物館 X

クリーンコンピュータ MZ-80C

シャープ MZ-80C

井戸禎光

Illustration TOSHIHITO ITO

1979年，シャープはCRT，カセットテープの一体となったセットで，ROM内のソフトの方は2KBのイニシアルプログラムローダ，それに2KBのキャラクタジェネレータのみの“クリーンコンピュータ”というコンセプトを提唱したMZ-80Cを268,000円で発表している．その特徴は，“ハードはインテグレーション済み，ソフトは買った人が自由に変えられる”ということであった．

CPUは8ビットのICで最も人気のあったZ-80である．Z-80はZILOG社が1986年に発表し，TRS-80（1977年発表）で使われて以来，多くの会社で使われることになるが，シャープでは1978年にキット（MZ-80K，198,000円）として発売し，IC自体の製造も行っている．このキットはメインボード，CRT，キーボード，カセットテープとばらばらの状態で買って，自分で組み立てるものであった．その組み立て完成品に当たるのがMZ-80Cであるが，キーボードの配列など多少の違いもある．

ソフトの方もクリーンだからといって，何も供給しなかったわけではなく，カセットテープ

を動かすモニターが付いていたので，カセット版でBASICやアセンブラをRAMに読み込ませてから使うという考え方であった．キャッチフレーズとしては，“ROM BASICから脱皮してアセンブラまたは高級言語に進もう——これはアマチュアからプロへの脱皮を意味する”というようなものであった．キーボード・カセット・音声・タイマーなどはソフトで制御している．

1979年という年は，日本のパソコンが“キットの時代”から本格的な“ワンボックスコンピュータ”の時代に移り変わるはじめの年として記念すべき年である．この移行をリードしたのがわが国の初期パソコン御三家である“PC-8001”“ベーシックマスター”それにこの“MZ-80C”なのである．

いずれも人気の高かったものであるが，またそれぞれ特徴が異なっており興味深いものがある．MZシリーズのハードウェアはモノクロCRTとカセットテープデッキを組み込んだ一体型の構成に特徴があった．フロッピーディスクは未だ高価であったので，別売り（298,000円）としていた．

このコンピュータはハードについてはケーブル1本つなぐのも億劫だが，ソフトについては腕に自信があるという類の人に喜ばれたのではないか．多くの機械が16KBまたは32KBしかRAMを積んでいないのに対し，48KBも積んでいた．このころのDRAMは4Kビットの4027とか8Kビットの4116といったものであったので，48KBも積むと結構ボード面積を食ってしまった．当時の機械のRAMは表示用に使ってしまう分も多く，ユーザーの使える分は10～15KBしかないものが多かった．一方，ROMの方は，当時BASICが書いてあり10KBASICとか少ないものが珍重されたり，逆に強力BASICといって24Kを必要とするものもあった．

キャラクタジェネレータには2316という2KBのROMを使い，1文字が8ドット×8＝8バイトとして8バイト×256文字に対応させている．文字表示用には4Kビットの2114を2個使って1000文字×8ビットとしている(残り24文字はあまり．漢字は使わないので文字として出てくるのは256種類＝8ビットで十分)．

次のMZ-80B2では指の力でスタート，ストップするメカ式のカセットテープデッキが電磁式になり，キーボードのファンクションキーの並びにテープ送りの押しボタンが付いた．一方，CRTを外してその代わりに小型のカラープリンタを一体化したMZ-700（1982年），カセットテープの代わりにクイックディスクを使ったMZ-1500(1982年)，さらにパソコンの画面とテレビの画面を重ね合わせて(スーパーインポーズ)ビデオに録画できる“パソコンテレビ X-1”(1982年)，カセットレコーダと2フロッピーのように見え，電話などの音声録音のできた“MZ-2531”と，ハードの方は新しい試みがどんどん製品化されていった．

その後（1987年ころまで）もシャープは高速のZ80B（6MHzなど）を用いた8ビット機の開発に力を入れていた．1987年になるとモトローラ系のCPUチップ68000を用いたX68000という新しい機種を発表している．最近では（1988年より），AX仕様のIBM PC/ATコンパチ機を作っている．

一方，そのほかの方面では，電卓はいうにおよばず，ディスプレイのシャープとしても日本の大きなシェアをとっている．それも単にCRTのみでなく，フラットパネルディスプレイの技術も高い．これもシャープペンシル以来，白黒TVの時代を経て，現在に至るまで“新アイデアと発明のシャープ”として面目躍如たるものがある．

シャープ MZ-80C I/O コネクタ信号図

番号	A列	B列
1	A15	GND
2	A14	＊INT
3	A13	GND
4	A12	＊MREQ
5	A11	GND
6	A10	＊IOREQ
7	A9	GND
8	A8	＊RD
9	A7	GND
10	A6	＊WR
11	A5	GND
12	A4	＊M1
13	A3	GND
14	A2	＊HALT
15	A1	GND
16	A0	RESET
17	GND	GND
18	D7	GND
19	D6	GND
20	D5	GND
21	D4	GND
22	D3	GND
23	D2	GND
24	D1	GND
25	D0	GND

Aは部品面，＊は負信号

祐安重夫

プログラマの道具学

第10回　ジャンクフード

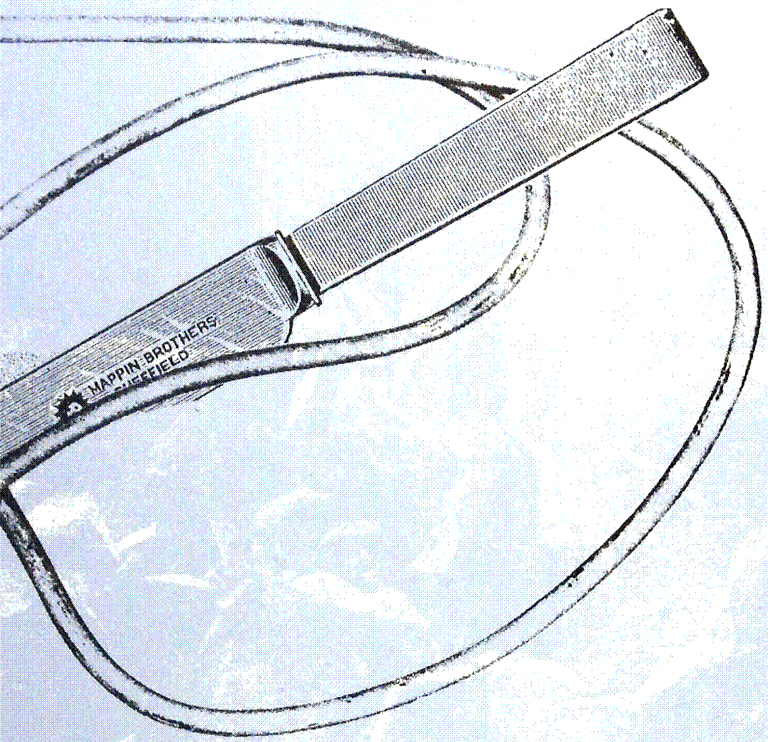
Illustration
Mikio Sasaki

サンドイッチ伯爵こと四代目ジョン・モンターギュ伯爵は，カード・ゲームをしながら食事ができるように，ローストビーフをパンの間にはさんでたべることをおもいつき，そのたべものはのちにサンドイッチと名付けられた．パンに肉をはさんでたべることが，サンドイッチ伯爵以前におこなわれていなかったはずはないのだが，レストランのメニューにも登場する正式の料理としてみとめられたのは，サンドイッチ伯爵がきっかけといえるだろう．

たしかにトーストしたイギリスパンに，上質の発酵バターとコールマンのマスタードでつくったマスタードバターをぬり，レアのローストビーフとクレソンをはさんだものは，絶妙のたべものである．サンドイッチは片手で手軽にたべられるものではあるが，仕事場でキーボードにむかいながらたべようとはおもわない．ひえたビールでもそえて，ゆっくり味わいたい．

プログラマの食事というと，キーボードにむかいながらたべるハンバーガやドーナツ，コンビニエンス・ストアのおにぎりといったイメージがつよいようだが，正直なところこれらの食物にあまり食欲をおぼえることはない．時間におわれてしかたなくたべてしまうことが，まったくないとはいえないのだが，普段はできるだけ自分で献立をかんがえ，自分で料理したものをたべるようにしている．そもそも仕事場で飲食することが，ほとんどない．

ハンバーガも一種のサンドイッチといえるが，大手のチェーンであつかっているような大量生産品は，味わってたべるよりはあわただしく腹につめこむのにむいているようだ．というより，じっくり味わっているとうんざりしてくるような気がする．結局キーボードにむかいながら，片手でたべるのがただしい



たべかたなのかもしれない．仕事場で食事をしないため，このようなジャンクフードとはあまり縁をもたずにすんでいる．

寝食をわすれてコンピュータにのめりこむことがないという点では，私自身はハッカーとはいえないのだろう．コンピュータにむかいながらジャンクフードをつめこむよりは，人生の中でかぎられた回数しかめぐってこない食事の時間を，できるかぎりたのしみたいとおもっているのである．

プログラマにとって自分で料理をつくることには，二つの意味がある．一つは仕事によるストレスを解消するためであり，もう一つはシステムとして料理を構築するたのしみのためである．別にプログラマにとってなどと，おおげさにかんがえることはない．単なる私の個人的な趣味である．

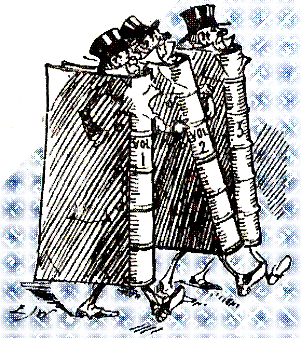
私の場合は最近ではこれに自然食志向という，やっかいなものがつけくわえられるようになった．無農薬または低農薬の野菜や果物の宅配サービスをうけはじめたので，それにあわせて一週間分の献立を構成するというのが，ここのところの習慣になっている．

プログラマのこのむジャンクフードの一種として，コーラなどのソフトドリンク類があり，最近では健康ドリンクへのこだわりもでてきているようだが，これなど自然食志向と類似しているようで，じつはかなりちがっているようでもある．健康ドリンクというのもなかなかうさんくさそうで，ためしてみたことはない．

などとえらそうなことをいっているが，私自身もジャンクフードとは無縁なわけではない．アメリカ風ジャンクフードには食指がうごかないが，日本風ジャンクフードにはじつはかなりこだわりがある．肉屋のコロッケには目がないし，仕事がたてこんでいる時の昼飯は，散歩がてら駅前まででて，立喰そばをたべる習慣になっている．

IC千夜一夜

第4回――初期のメモリ素子



――遅延線メモリとコアメモリ――

虎岩 登

今回はメモリ，とくにIC化される前の古い技術について見てみましょう．

現在，広く用いられているノイマン型コンピュータの特徴は，作業手順をあらかじめメモリの中に入れておき，CPUがそれを順に読みだして実行するところにあります．複雑な作業では，比較的単純な命令を大量に組み合わせてメモリに記憶させてから，実行します．したがって，コンピュータシステムのハードウェア能力は，CPUの能力や周辺機器の性能だけではなく，利用可能なメモリ量によっても大きく左右されます．コンピュータが誕生して以来，メモリ量を増やし続けているのも，そこに理由があります．

コンピュータが開発されてから現在にいたるまで，真空管，パラメトロン，CRT，水銀遅延線などの遅延線，磁気コア，磁気ドラム，トランジスタなど，さまざまなものが主記憶用のメモリとして用いられてきました．今回は，この中から遅延線メモリとコアメモリを取り上げて，その動作原理を眺めてみることにしましょう．

ダイナミック・シーケンシャルメモリ（D-SAM）

遅延線メモリ

図1は，遅延線メモリのモデルです．山に登るとこだまがよく聞こえる場所がありますね．そこにマイクと大出力のアンプ，スピーカーをセットしたとしましょう．これらのマイクからスピーカーまでの装置には，記憶作用はないことに注意してください．また，スピーカーの出す音を直接マイクが拾わないようにしておきます．準備ができたらマイクに向かって「やっほー」と叫んでみます．スピーカーから向かいの山に向かって同じ声が発されて，しばらくするとこだまが戻ってきます．それをマイクが拾います．ボリュームを適切に調整しておけば，スピーカーからまた同じ大きさの声で「やっほー」と声が出ていきます．何回，何十回とマイクースピーカーと向かいの山の間を「やっほー」が往復して，ちょうど録音されたのと同じような状態になります．これは音声がメモリされた状態といえるでしょう．

いったい，このモデルのどこに本質的な記憶作用があったのでしょうか．はじめに断っておいたように，マイクやアンプ，スピーカーには記憶作用はありません．そうすると，残るはスピーカーからマイクに戻るまでの「空間」ということになります．「空間」が，スピーカーから出た音が空気中を伝わっている間，音を保持していたというわ

けです．この伝わっている間の「遅れ」が，記憶作用の本質だといえるでしょう．秒単位のわずかな時間ではあるけれど，音を保持する，つまり記憶しておくことができさえすれば，すぐさまそれを投げ返すことで，永続的な記憶が可能になります．

基本的にわずかな時間しか記憶できない素子で永続的な記憶を可能にするため，記憶が消えてしまう前に再書き込みするようなメモリをダイナミックメモリと呼びます．したがって，ここで説明した方式もダイナミックメモリの一種になります．

実際のコンピュータでは，遅延線という素子を使います．これは入力信号を一定の時間だけ遅れて出力するような素子です．記憶素子としては，水銀の中に特殊なマイクとスピーカの役をするトランスデューサを入れて，超音波振動の形で信号を遅らせる水銀遅延線がおもに使われました．やはりある種のこだまを使っていたわけですね．

図1　こだまによる音声記憶モデル

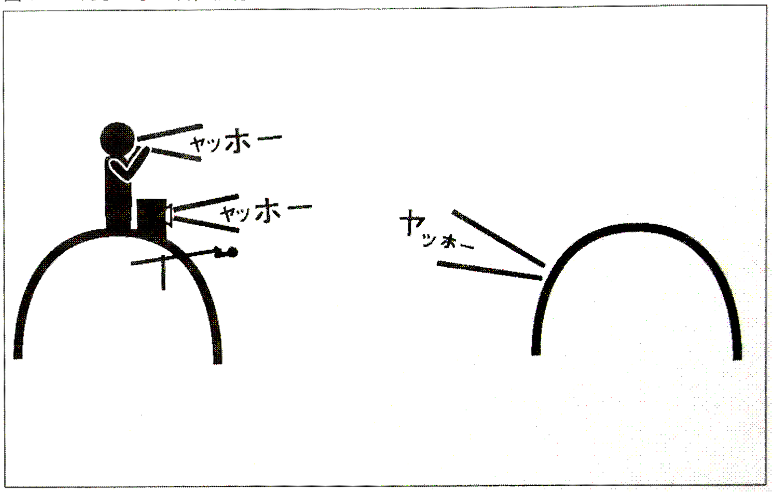
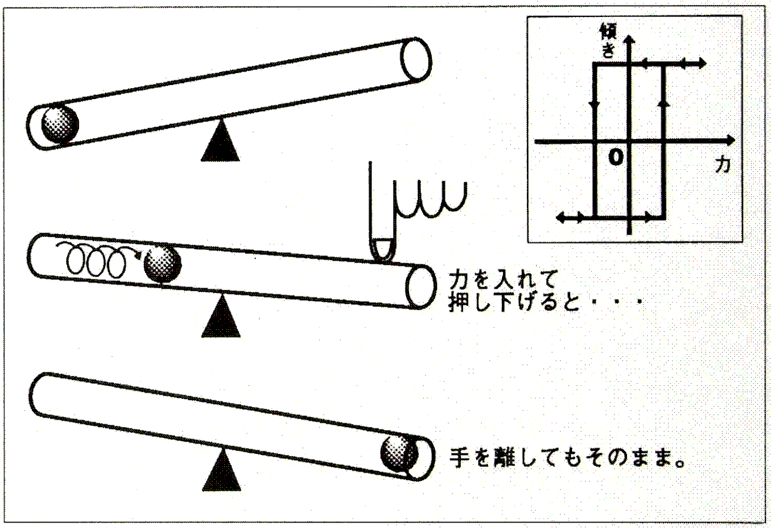


図2　双安定性を示す力学モデル



スタティック・ランダムアクセスメモリ（S-RAM）

コアメモリ

スタティックメモリの原理を表す力学的モデルを図2に示します．両端が閉じたパイプを使ってシーソーを作ります．パイプの中には鉄の球が入っていて，自由にころがるようになっています．こうすると，鉄球のある側が下になって静止していますが，反対側を手で押し下げれば鉄球がころがってきて手を離しても元には戻りません．このような性質を「双安定性」と呼びます．2つの安定な状態があって，一方の状態から勝手にもう一方の状態に変化することはない，無理に変化させれば，もう一度状態を変えてやるまでは，そのままになっているという性質です．したがって，双安定性を持つ素子や回路とは，メモリそのものといえるわけです．この双安定性を利用したメモリがスタティックメモリです．スタティックメモリでは，ダイナミックメモリのように内部で繰り返し書き込み動作を行う必要はありません．

コアメモリは，強磁性体をドーナツのように固めたもので，ドーナツの穴に電線を入れて使用します．電線を流れる電流とドーナツ内部の磁束密度（磁石の強さといった意味です）のグラフを図3に示します．電流がある量を超えると，急激にその電流の発生する磁界と同じ向きにコア内部の磁力が変化して，電流を0に戻しても，その向きの磁力がコア内部に残ります．逆向きの電流を同じように流すと，その電流を取り除いてもコア内部の磁力は先ほどとは逆向きになって残ります．つまり，シーソーモデルと同じ双安定性を持っていることがわかります．

アクセス方式

SAMとRAMの違い

次に，アクセス方法からこの2種類のメモリを比較してみましょう．結論からいえば，遅延線メモリはシーケンシャルアクセスで，コアメモリはラン

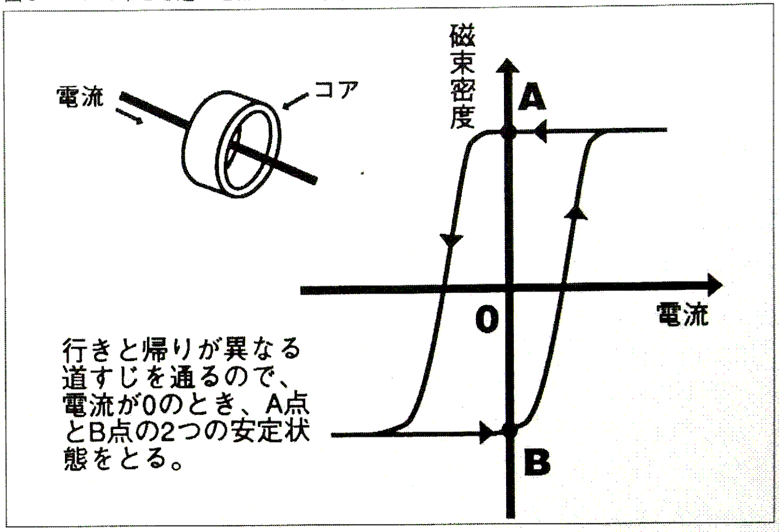
ダムアクセスに分類されます．シーケンシャルアクセスとランダムアクセスというと，ファイルのデータをアクセスする技法だと思っておられる方も多いでしょう．コンピュータのメインメモリは，ランダムアクセスがあたりまえになっているのですが，実際にランダムアクセスが可能なメモリを設計するのはたいへんなのです．

シーケンシャルアクセスだけなら，読みだし／書き込みを行う回路は1つだけで，あとはバケツリレー式にデータを隣に順次渡していくだけですみます．データの入り口と出口は1か所ずつでよく，データのどれにアクセスするかは，タイミングによって決めればよいことになります．

ランダムアクセスを実現するには，データの切り替えスイッチが必要です．でも，図4 (b) のように単純に切り替えスイッチを並べてしまうと，1 Mbitメモリのときには100万個以上のスイッチが必要になります．これではデータを記憶するところよりスイッチ回路のほうが大きくなってしまうので，一般的には2次元的に記憶素子を配置し，スイッチを2段重ねにして数を減らします．こうすれば，1Mbit = 1024×1024bitですからスイッチは1024+1024で2048個で済みますが，それでもこの切り替えスイッチの分だけ複雑になるのは間違いないでしょう．

では，シーケンシャルアクセスメモリ (SAM) の代表，遅延線メモリからみていきましょう．遅延線の中をデータが1周して出てくる時間を遅延時間といいますが，たとえばこの遅延時間を100等分して単位時間毎に0か1かを記録することにすれば，100bitのメモリになります．遅延線の外側にカウンタを用意して遅延時間の100分の1単位の時計を作れば，その時計が示す

図3　コアの中心を通る電流とコア内部の磁束密度の関係



時刻がアドレス情報として使えます．カウンタが3を示していれば3番地のデータが出力されて再書き込みするところ，56なら56番地が同じく出入り口にある状態と解釈するわけです．こうすれば1本の遅延線で多くのビット数のデータを記憶できます．ただし，特定のアドレスのデータを読み出そうとしても，カウンタがちょうどそのアドレスを示していなければデータは得られません．遅延線の中から目指すデータが出てくるまで，ずっと待たなければならないのです．

欠点はあったにせよ，このようなメモリが実際に使われていました．基本的にCPUは分岐命令がなければメモリを順番にアクセスしますから，次に出てくるデータに対するSAMのアクセス時間と，CPUの命令実行時間を一致させておけば，分岐命令を実行するときだけCPUをウエイトさせればすみます．あとはまったく時間の無駄がありません（もっともウエイト時間は非常に長いのですが）．データをアクセスするときでも，テーブルを順番に検索したり，データの総和を計算するような場合にはSAMでもあまり効率は落ちません．むしろ，1本の遅延線と簡単なハードウェアで，比較的多くの記憶ができた点が評価された時代があったのです．

これに対して，コアメモリはランダムアクセスメモリ (RAM) の古典的代表格です．1ビットを記憶するのに1個のコアが必要で，ハードウェアは複雑かつ高価になりますが，システムの一部としては使いやすいメモリです．

コアメモリは，たいへん興味深い方法でRAMを実現しています．図4ではコアの中には1本の電線しか通っていませんが，ふつうは横糸と縦糸のように上下左右に通る線と，すべてのコアに斜めに通っている1本の合計3本の線が通っています．実現方法にはいくつかのバリエーションがあるのですが，ここでは簡略化して説明します．

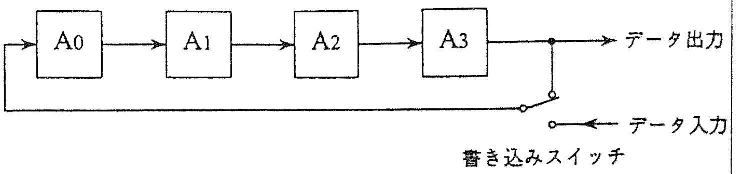
まずは書き込み方です．縦と横の線には，それぞれコアの磁界を反転させることのできる電流の1/2より少し大きめの電流が，必要に応じて流せるようになっています．何本もある縦線と横線のうち，それぞれ1本ずつに電流を流してやると，縦横の電線の交点にあるコアだけが2本分の影響を受けますから，磁界が反転します．ほかのコアは，反転に十分なだけの電流が流れ

ませんから影響を受けません．これで特定のコアだけに書き込むことができました．

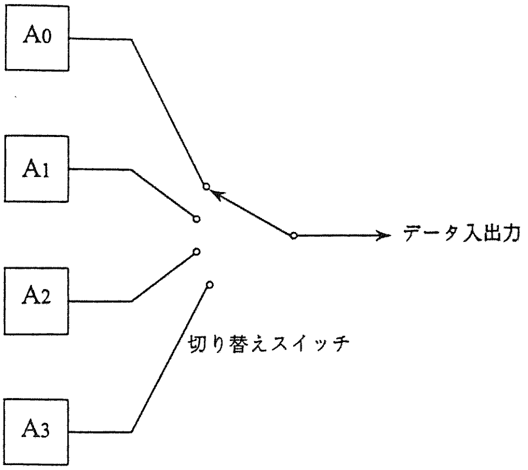
読み出し方も巧妙です．最初に取り合えず目的の番地のコアの磁界を，たとえば，０を意味する向きに向けるように書き込んでみます．ここで第３の線，つまりすべてのコアに通っている線が役に立ちます．もしも目的のコアに１が書き込んであったなら，磁界が反転することになります．そうすると電磁誘導の法則によって，コアの中を通る電線には逆起電力が発生します．ところが，仮に最初から目的のコアには０が書き込んであったなら，磁界は変化しませんから，特に大きな逆起電力は発生しません．したがって，第３の線の電圧を増幅すれば，０か１か，どちらが書き込まれていたかがわかります．

でも，これでは読み出した後の記憶内容は消されてしまっていますね．これを専門用語では破壊読み出しといいます．１回しか読み出せないメモリというのは困りますから，読み出し動作を行った直後に自動的に読み取ったデータを書き込むような回路が作り込まれています．このようなメモリにはアクセスタイムとサイクルタイムが一致しないとか，リード・モデファイ・ライト動作が可能であるなどの特徴がありますが，これらの言葉の意味するところは来月説明することにします．現在の主流であるLSIによるD-RAMも破壊読み出しが行われていて，同じ特徴を持つのです．

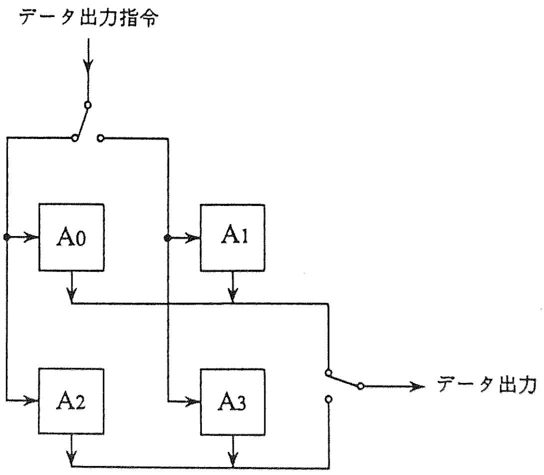
図４　切り替えスイッチの配置



(a) シーケンシャルメモリのデータ選択法
メモリの内部をデータが順に流れているので、１ヶ所で目的のデータが出てくるのを待ちかまえる。



(b) ランダムアクセスメモリのデータ選択の原理
記憶場所ごとにアドレス選択用のスイッチが接続されている



(c) 実際のランダムアクセスメモリのデータ選択法
２次元的（２段階）選択を行うことでスイッチの数を減らしている

MS-DOS TOOL LABO

MS-DOSツール・ラボ

第7回 FAST基本ライブラリ"コア"
FORTHコンパイラの作成③ FAST

加野島英渡

先月作成したFAST.OBJに基本ライブラリ"コア"を加えることで，FORTHコンパイラFASTを完成させる．基本ライブラリの解説に加えて，FASTプログラミングの方法についても解説を行った．

コア(基本ライブラリ)

■コアの概要

コアはFASTの基本ライブラリである．コンパイラ自身もこれにリンクして動作している．現在はまだ整備が行き届いておらず，コンパイラの記述に必要な程度の最低限のメモリアクセスやスタック操作ワードなどが定義されているのみである．今後読者自身でコアを充実させていってもらいたい．

今回実現したバージョンでは，スタックサイズを2ワードにした．演算などは速度を重視して下位の1ワードを対象に行うが，メモリアクセスワードではフルアドレスを意識してライブラリを構成している．この場合のセグメント値の格納のために上位ワードを使うのである．

■スタック仕様の見方

FORTH系言語ではスタックレベルの管理が重要である．うっかりしてスタックにデータを積んだままにしたり，逆にスタックにデータがないのにさらに読み出すような処理をしたりすると，プログラムが正常に動作しなくなる可能性がある．それがループの中だったりするとスタックオーバーフローやアンダーフローを起こしてプログラムが暴走する危険さえある．

したがって，ワードには必ずそれを呼び出す前と戻ってきたときのスタックの状態をコメントとして書いておくことを強くお勧めする．FASTでは図1のような書式を標準とする．

'|'の左側がワードへの引数，右側が戻り値である．いずれもスタック上のデータの置かれ方を表わしている．データは右端がスタックトップである．

図1の例では，このワードが文字列

ワンラインNEWS
OA研究所など3社は，電波に比べ外部ノイズに強い赤外線を使った通信モデム「IRM96」を開発，パソコン同士やプリンタをワイヤレスで結ぶことができる．

(str)，バッファアドレス(buf)の順にデータを積んで呼び出され（バッファアドレスのほうがスタックトップ），この2つのデータを消費して結果をなにも返さないという意味である．

図1

```
[ WORD_NAME    ; str,buf | . ]
```

内部処理・データ構造など

■文字列の取り扱い

FASTでは文字列をその先頭アドレスで扱う．文字列の本体をメモリ空

表1 コアのワード一覧

<table>
<tr><td>exit</td><td>d | .</td><td>終了コードを返すプログラムの終了</td><td>s=</td><td>str,str | T/F</td><td>文字列比較</td></tr>
<tr><td>exit0</td><td>. | .</td><td>終了コード0を返すプログラムの終了</td><td>stail</td><td>str | addr</td><td>文字列の末尾アドレスを取得</td></tr>
<tr><td>$$startup</td><td>. | .</td><td>【ライブラリ使用前に必ず呼び出すこと】</td><td>c@stail</td><td>addr | d</td><td>文字列の最後の文字を取得</td></tr>
<tr><td>psp_seg</td><td>. | seg</td><td>プログラム起動時のPASセグメント値</td><td>slen</td><td>addr | d</td><td>文字列のバイト数を取得</td></tr>
<tr><td>my_psp</td><td>. | addr</td><td>データセグメント内の疑似PSPアドレス</td><td>cupcase</td><td>d | D</td><td>小文字を大文字に変換</td></tr>
<tr><td>pstack</td><td>. | d</td><td>パラメータスタックの値</td><td>+</td><td>3,2 | 5</td><td>加算</td></tr>
<tr><td>?pstack_initial</td><td>. | T/F</td><td>パラメータスタックの空を確認する</td><td>1+</td><td>1 | 2</td><td>インクリメント</td></tr>
<tr><td>$$wnumber</td><td>. | d</td><td>コンパイラが呼び出す特殊なワード</td><td>2+</td><td>1 | 3</td><td>2だけ増加</td></tr>
<tr><td>$$cstring</td><td>. | str</td><td>//</td><td>-</td><td>3,2 | 1</td><td>減算</td></tr>
<tr><td>$$jmptrue</td><td>d | .</td><td>//</td><td>1-</td><td>3 | 2</td><td>デクリメント</td></tr>
<tr><td>$$jmpfalse</td><td>d | .</td><td>//</td><td>2-</td><td>3 | 1</td><td>2だけ減少</td></tr>
<tr><td>$$do_entry</td><td>d | .</td><td>//</td><td>*</td><td>6,2 | 12</td><td>乗算</td></tr>
<tr><td>$$loop_entry</td><td>. | .</td><td>//</td><td>/</td><td>6,2 | 3</td><td>除算</td></tr>
<tr><td>leave</td><td>. | .</td><td>DO〜LOOPの脱出，ループ回数をカウンタにコピー</td><td>mod</td><td>7,2 | 1</td><td>剰余</td></tr>
<tr><td>i</td><td>. | d</td><td>リターンスタックのトップを参照</td><td>/mod</td><td>7,2 + 1,3</td><td>商と剰余</td></tr>
<tr><td>j</td><td>. | d</td><td>リターンスタックの2番めを参照</td><td>0<</td><td>-1 | T</td><td>0より小さいか？</td></tr>
<tr><td>r@</td><td>. | d</td><td>iと同じ</td><td>0></td><td>1 | T/F</td><td>0より大きいか？</td></tr>
<tr><td>r></td><td>. | d</td><td>リターンスタックトップをスタックに移動</td><td>0=</td><td>0 | T/F</td><td>0であるか？</td></tr>
<tr><td>>r</td><td>d | .</td><td>スタックトップをリターンスタックに移動</td><td>0<></td><td>0 | T/F</td><td>0でないか？</td></tr>
<tr><td>rdrop</td><td>(d) | (.)</td><td>リターンスタックのデータを1つ捨てる</td><td>=</td><td>2,2 | (TRUE)</td><td>等しいか？</td></tr>
<tr><td>return</td><td>. | .</td><td>上位ワードに戻る，DO〜LOOPでは使用できない</td><td><></td><td>2,2 | F</td><td>等しくないか？</td></tr>
<tr><td>dup</td><td>d | d,d</td><td>スタックトップを複写</td><td>></td><td>3,2 | (TRUE)</td><td>大きいか？</td></tr>
<tr><td>?dup</td><td>d | d,d / d</td><td>スタックトップが真（ゼロ以外）なら複写</td><td><</td><td>3,2 | F</td><td>小さいか？</td></tr>
<tr><td>drop</td><td>d | .</td><td>スタックトップのデータを1つ捨てる</td><td>>>></td><td>4 | 8</td><td>1ビット右シフト</td></tr>
<tr><td>2drop</td><td>d,d | .</td><td>スタックデータをトップから2つ捨てる</td><td>>></td><td>4,2 | 16</td><td>右シフト</td></tr>
<tr><td>swap</td><td>d1,d2 | d2,d1</td><td>スタックトップと2番目を交換</td><td><<<</td><td>4 | 2</td><td>1ビット左シフト</td></tr>
<tr><td>over</td><td>d1,d2 | d1,d2</td><td>スタックの2番目のデータを複写</td><td><<</td><td>16,2 | 4</td><td>左シフト</td></tr>
<tr><td>@</td><td>addr | d</td><td>アドレスからワード値を読み出す</td><td>&</td><td>d,d | d</td><td>ビット演算AND</td></tr>
<tr><td>c@</td><td>addr | d</td><td>アドレスからバイト値を読み出す</td><td>|</td><td>d,d | d</td><td>ビット演算OR</td></tr>
<tr><td>1+@</td><td>addr | d</td><td>アドレスのワード値をインクリメントし読み出す</td><td>&&</td><td>T/F,T/F | T/F</td><td>論理演算AND</td></tr>
<tr><td>@1+</td><td>addr | d</td><td>アドレスのワード値を読み出し後インクリメント</td><td>||</td><td>T/F,T/F | T/F</td><td>論理演算OR</td></tr>
<tr><td>!</td><td>d , addr | .</td><td>ワードデータをアドレスにストア</td><td><range></td><td>d,L,H | T/F</td><td>範囲内か？</td></tr>
<tr><td>c!</td><td>d , addr | .</td><td>バイトデータをアドレスにストア</td><td>>range<</td><td>d,L,H | T/F</td><td>範囲外か？</td></tr>
<tr><td>+!</td><td>d , addr | .</td><td>アドレスのワードデータを増加</td><td>sprint_handle</td><td>addr,h | .</td><td>ハンドル指定の文字列表示</td></tr>
<tr><td>1+!</td><td>addr | .</td><td>アドレスのワードデータをインクリメント</td><td>sprint</td><td>addr | .</td><td>文字列表示</td></tr>
<tr><td>-!</td><td>d , addr | .</td><td>アドレスのワードデータを減少</td><td>cprint_handle</td><td>d , h | .</td><td>ハンドル指定の文字表示</td></tr>
<tr><td>1-!</td><td>addr | .</td><td>アドレスのワードデータをデクリメント</td><td>cprint</td><td>d | .</td><td>文字表示</td></tr>
<tr><td>full_addr</td><td>off,seg | addr</td><td>segをoffの上位ワードに書き込む</td><td>crlf_handle</td><td>handle | .</td><td>ハンドル指定の改行</td></tr>
<tr><td>reg_ds</td><td>. | seg</td><td>セグメントレジスタ値を取得</td><td>crlf</td><td>. | .</td><td>改行</td></tr>
<tr><td>reg_cs</td><td>. | seg</td><td>//</td><td>key_wait</td><td>. | .</td><td>キー入力待ち</td></tr>
<tr><td>reg_ss</td><td>. | seg</td><td>//</td><td>cinput</td><td>. | d</td><td>エコーなしキー入力</td></tr>
<tr><td>set_reg_ds</td><td>d | d</td><td>セグメントレジスタ値を上位ワードに設定</td><td>cinpute</td><td>. | d</td><td>エコーつきキー入力</td></tr>
<tr><td>set_reg_cs</td><td>d | d</td><td>//</td><td>fcinput</td><td>h | d</td><td>ハンドル指定の文字入力</td></tr>
<tr><td>set_reg_ss</td><td>d | d</td><td>//</td><td>fsinput</td><td>buf , h | T/F</td><td>ハンドル指定の一行入力</td></tr>
<tr><td>c+</td><td>d,str | .</td><td>文字を文字列に追加</td><td>sinput</td><td>buf | T/F</td><td>一行入力</td></tr>
<tr><td>s!</td><td>str,buf | .</td><td>文字列をバッファにコピー</td><td>fopen</td><td>fn,m | h,T/F</td><td>ファイルオープン</td></tr>
<tr><td>s+</td><td>str,buf | .</td><td>文字列をバッファに追加</td><td>fclose</td><td>h | T/F</td><td>ファイルクローズ</td></tr>
</table>

凡例：

- addr : address
- buf : buffer
- d : data
- off : offset
- seg : segment
- str : string
- h : file handle
- fn : filename
- T/F : TRUE or FALSE
- . : no data
- L : lower limit
- H : higher limit
- m : mode
- () : data (on return stack)

間のどこに置いてもいいように，上位ワードにセグメント値が入ったフルアドレスとしている．なお，文字列の本体はASCIZ文字列（0で終端された不定長の文字並び）である．

■パラメータスタックのメモリイメージ

下位，上位と書いているが，先月号に書いたようにFASTのパラメータスタックはBPレジスタをスタックポインタとして，スタックセグメントのオフセット0から増加方向に使用する（図2）．この取り扱いがちょっと特殊なので改めて説明しておきたい．

パラメータスタックポインタは初期状態ではパラメータスタック領域の下限からさらに2バイト下（図2では上）を指している．パラメータスタック領域はスタックセグメントのオフセット0からはじまるので，パラメータスタックポインタの初期値は－2である．

パラメータスタックサイズは2ワードであるから，スタックにデータを積む場合はパラメータスタックポインタを4バイト増加させることになる．このところを詳しく書くと図3のようになる．図2と違って1つの枠が1バイトを表わしているので，注意して見ていただきたい．

この状態でパラメータスタックポインタの位置を下位ワード，そこから2バイト戻ったところを上位ワードとして扱うのだが，下位ワードは（＋2）と（＋3）に書き込まれることになる．つまりパラメータスタックポインタが指しているアドレスの1バイト先までが使われることになるので，この領域を本来のパラメータスタック以外の目的で使用する場合は注意していただきたい．

■スタートアップ処理

“$$startup”というワードについては，とくに説明しておかなければならない．このワードは，起動時のPSPセグメントをライブラリ内部の変数に保存し，PSPセグメントから自分自身の管理するデータセグメントに100Hバイトをコピーする．これによってコマンドラインオプションなどの参照を容易にしている．また，パラメータスタックポインタとして使用するBPレジスタの初期化もこの中で実行している．

図2

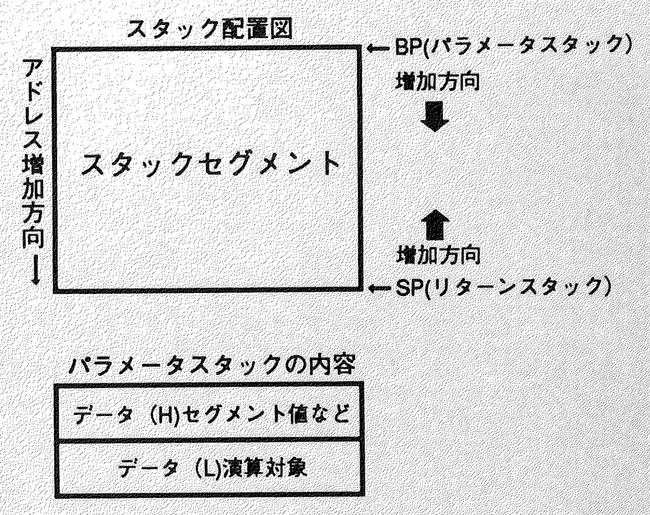
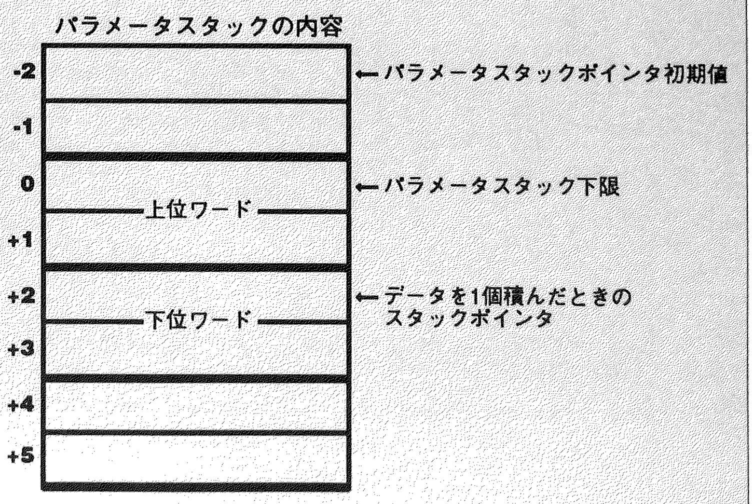


図3



ワンラインNEWS
九州松下電器は，ISDNにパソコンなどの端末を回線接続するターミナルアダプタの低価格製品を発売．価格は148,000円で，NTTのISDN「INSネット64」に対応．

リスト1　CORE. F

```
TITLE   FAST Runtime libraly

; ========================================================
;
; 【FASTコンパイラ】　ランタイムライブラリ
;
;       Copyright(C) 1991 By Justech Labo. H.Kanoshima
;
; ========================================================
;
; 1991.04.28    : Create

; 1991.05.31    : Fix
;

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT , DS:_DATA


; ========================================================
;
;                       定数宣言など
;
; ========================================================

FALSE           equ     0
TRUE            equ     1

STDIN           equ     0
STDOUT          equ     1

INITIAL_BP      equ     0ffffh

CR              equ     0dh
LF              equ     0ah
EOF             equ     1ah

RBUFSIZ         equ     512

; ========================================================
;
;                       システムワード
;
; ========================================================

[ exit              ; d | .         ;終了コードを返すプログラムの終了
*       mov     al,[bp]
*       sub     bp,4
*       mov     ah,4ch
*       int     21h
]

[ exit0             ; . | .         ;終了コード0を返すプログラムの終了
*       add     bp,4
*       mov     word ptr [bp],0
        exit
]

[ $$startup         ; . | .         ;【ライブラリ使用前に必ず呼び出すこと】
*       mov     ax,_DATA
*       mov     es,ax
*       mov     es:pspbuf,ds
*       mov     cx,256
*       mov     si,0
*       mov     di,offset newpsp
*rep    movsb
*       mov     ds,ax
*       mov     word ptr [rboff],0
*       mov     bp,INITIAL_BP
*       pop     bx
*       mov     ax,offset f_exit0
*       push    ax
*       jmp     bx
]

[ psp_seg           ; . | seg       ;プログラム起動時のPSPセグメント値
*       mov     ax,pspbuf
*       add     bp,4
*       mov     [bp],ax
]

[ my_psp            ; . | addr      ;データセグメント内の疑似PSPアドレス
*       mov     ax,offset newpsp
*       add     bp,4
*       mov     [bp - 2],ds
*       mov     [bp],ax
]

[ pstack            ; . | d         ;パラメータスタックの値
*       mov     ax,bp
*       add     bp,4
*       mov     [bp],ax
*       mov     [bp - 2],ss
]

[ ?pstack_initial   ; . | T/F       ;パラメータスタックの空を確認する
        pstack (INITIAL_BP) =
]

; ========================================================
;
;                       暗黙に呼び出される特殊なワード
;
; ========================================================
;
;自分を呼び出したコードの直後のワード値をスタックに積む
;
; ======================
;       call    $$wnumber
;       dw      nnnn
; ======================
;
[ $$wnumber         ; . | d         ;コンパイラが呼び出す特殊なワード
*       pop     bx
*       add     bp,4
*       mov     ax,cs:[bx]
*       mov     [bp],ax
*       mov     [bp - 2],ds
*       add     bx,2
*       jmp     bx
]
;
;自分を呼び出したコードの直後の文字列をスタックに積む
;
; ======================
;       call    $$cstring
;       dw      6
;       db      "12345",0
; ======================
;
[ $$cstring         ; . | str       ;コンパイラが呼び出す特殊なワード
*       add     bp,4
*       mov     [bp - 2],cs
*       pop     di
*       mov     bx,di
*       inc     di
*       inc     di
*       mov     [bp],di
*       add     di,cs:[bx]
*       jmp     di
]
;
;スタックトップが真ならば自分を呼び出したコードの直後にあるオフセット値に飛ぶ
;
; ======================
;       call    $$jmptrue
;       dw      label
; ======================
;
[ $$jmptrue         ; d | .         ;コンパイラが呼び出す特殊なワード
*       mov     ax,[bp]
*       sub     bp,4
*       pop     bx
*       or      ax,ax
*       jnz     _jt1
*       add     bx,2
*       jmp     bx
*_jt1:
*       jmp     cs:[bx]
]
;
;スタックトップが偽ならば自分を呼び出したコードの直後にあるオフセット値に飛ぶ
;
; ======================
;       call    $$jmpfalse
;       dw      label
; ======================
;
[ $$jmpfalse        ; d | .         ;コンパイラが呼び出す特殊なワード
*       mov     ax,[bp]
*       sub     bp,4
*       pop     bx
*       or      ax,ax
*       jz      _jf1
*       add     bx,2
*       jmp     bx
*_jf1:
*       jmp     cs:[bx]
]
;
;DOの実体。
;ループカウンタ(初期値は0)とパラメータスタック上のループ回数をリターン
;スタック上に積む。
;
; ======================
;       call    $$do_entry
;label:
; ======================
;
[ $$do_entry        ; d | .         ;コンパイラが呼び出す特殊なワード
*       pop     bx
*       mov     ax,[bp - 2]     ; terminate    H
*       push    ax
*       mov     ax,[bp]         ; terminate    L
*       push    ax
*       xor     ax,ax
*       push    ax              ; loop counter H
*       push    ax              ;              L
*       sub     bp,4
*       jmp     bx
]
;
;LOOPの実体。
;ループカウンタとループ回数を比較し、回数に達していなければ自分を呼び出した
;コードの直後にあるオフセット値に飛ぶ。(これは直近のDOの位置)
;
; ========================
;       call    $$loop_entry
;       dw      label
; ========================
;
[ $$loop_entry      ; . | .         ;コンパイラが呼び出す特殊なワード
*       pop     bx
*       push    bp
*       mov     bp,sp
*       mov     ax,[bp + 2]     ; loop counter L
*       add     ax,1
*       jc      _le1
*       cmp     ax,[bp + 6]     ; terminate    L
*       jae     _le1
*       mov     [bp + 2],ax
*       pop     bp
```

ワンラインNEWS
セイコーエプソンは、パソコン用プリンタとしては業界初の4種類の用紙を常時セットしワンタッチで選択できる漢字プリンタ「VP1700」と「VP870」を発売した。

プログラマはプログラムの第1ステップとして必ずこのワードを呼び出さなくてはならない．これを怠るとコアが正常に動作できない．最初に実行するワードの先頭でこのワードを呼び出すようにしておけばいい．

なお，最初に実行するワードの名前は任意だが，ラベル名の先頭には必ず“f_”がつくことを忘れないようにしていただきたい．たとえば最初に実行するワードの名前を“main”とすると，プログラムの最後に

```
end  f_main
```

という記述が必要である（サンプルコーディングを参照のこと）．

FASTプログラミングの実際

■サンプルコーディング

では，サンプルコーディングをとおしてFASTのプログラミングを解説しよう．ありきたりだが，歓迎のメッセージと0～9までの10個の数値を表示するというものである（リスト2）．予備知識として以下のようなことをおぼえておいていただく．

(1)FASTは実行開始時にはアセンブラモード（素通し）になっている．

(2)ソースファイルの拡張子はなんでもいいが，慣例で“.F”をつけることになっている．

(3)“;”以降はコメントとして扱われ，オブジェクト生成には意味をもたない．

(4)FAST記述の開始は，ソース行先頭の‘[’である．1個以上の空白を置いてワード名を書き，スタック仕様を記述する．スタック仕様はコメントとして書かれているが，コンパイラのソースを見ていただけばわかるように，実際にはワード名以降は無視されているのでなにをどのように書いてもいい．これは，ワード名のあと（おなじ行）に処理を書いても無視されるということでもある．

(5)FAST記述の終了は，任意の位置に現われる“]”というワードである．このワードのためのオブジェクトを生成したあと，FASTはまたアセンブラモードに戻る．

(6)ワード名は内部でラベル名に変換されるので（単語の認識の解説を参照のこと），アセンブラモードからFAST記述のワードを参照する場合にはラベル名を使わなければならない．プログラムの最後が

```
END  main
```

ではなく

```
END  f_main
```

となっているのはそのためである．

セグメント定義などはすべてアセンブラそのものである．コアとリンクするのでセグメント名などはコアと合わせておかなくてはならない．もしこれ以外の構成にするのならば，コアのほうをそれに合わせて作り直せばいい．

“_print_count”というワードは，スタック仕様でわかるように，スタックから数値を1つとり，それに‘0’，つまり30Hを加えて表示するものである．これに0～9の数値を与えると‘0’から‘9’という文字を表示してくれる．このワードをスタックの動きとともに少し詳しく追ってみよう．

まず‘0’は数値として扱われるので，30Hがスタックに積まれる．このワードが呼び出されるときには，上位のワードですでに数値が1つ積まれているので，スタックには2個のデータが積まれることになる．

次に‘+’はスタックから2個のデータを取り出して加算し，結果をスタックに積むワードであるから，ここでスタック上のデータは加算結果1個だけになる．スタックからデータを取り出すと，そのデータはスタック上から消滅してしまうことを思い出していただきたい．

“cprint”はスタックから1個のデータを取り出し，それを標準出力に送るワードである．これでスタックが空になる．

リスト2　TEST.F

```
include         core.h

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT , DS:_DATA

[ _print_count          ; d | .
        '0' + cprint CRLF ]

[ main
        $$startup
        "Welcome to FAST" sprint crlf
        10 do
                i _print_count
        loop
]

_TEXT   ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'

_DATA   ENDS

        END     f_main

```


ワンラインNEWS
ジャストシステムは，AXパソコンに対応したワープロソフト「一太郎VER.4」を発売した．日立製作所「FLORA」，三菱電機「MAXY」で使える．価格は68,000円

```
*        jmp     cs:[bx]
*_lel:
*        pop     bp
*        add     bx,2
*        add     sp,8
*        jmp     bx
]

; ==========================================================
;
;                       スタック操作用のワード
;
; ==========================================================

[ leave              ; . | .        ;DO~LOOPの脱出。ループ回数をカウンタにコピー
*        mov     bx,sp
*        mov     ax,ss:[bx + 6]
*        mov     ss:[bx + 2],ax
*        mov     ax,ss:[bx + 8]
*        mov     ss:[bx + 4],ax
]

[ i                  ; . | d        ;リターンスタックのトップを参照
*        add     bp,4
*        mov     bx,sp
*        mov     ax,ss:[bx + 2]
*        mov     [bp],ax
*        mov     ax,ss:[bx + 4]
*        mov     [bp - 2],ax
]

[ j                  ; . | d        ;リターンスタックの二番めを参照
*        add     bp,4
*        mov     bx,sp
*        mov     ax,ss:[bx + 6]
*        mov     [bp],ax
*        mov     ax,ss:[bx + 8]
*        mov     [bp - 2],ax
]

[ r@                 ; . | d        ;i とおなじ
*        jmp     f_i
]

[ r>                 ; . | d        ;リターンスタックトップをスタックに移動
*        pop     bx
*        add     bp,4
*        pop     ax
*        mov     [bp],ax
*        pop     ax
*        mov     [bp - 2],ax
*        jmp     bx
]

[ >r                 ; d | .        ;スタックトップをリターンスタックに移動
*        pop     bx
*        mov     ax,[bp - 2]
*        push    ax
*        mov     ax,[bp]
*        push    ax
*        sub     bp,4
*        jmp     bx
]

[ rdrop              ; (d) | (.)    ;リターンスタックのデータをひとつ捨てる
*        pop     bx
*        add     sp,4
*        jmp     bx
]

[ return             ; . | .        ;上位ワードに戻る。DO~LOOPでは使用できない
*        add     sp,2
]

[ dup                ; d | d,d      ;スタックトップを複写
*        add     bp,4
*        mov     ax,[bp - 4]
*        mov     [bp],ax
*        mov     ax,[bp - 6]
*        mov     [bp - 2],ax
]

[ ?dup               ; d | d,d / d ;スタックトップが真（ゼロ以外）なら複写
*        cmp     word ptr [bp],FALSE
*        jz      _?dpl
*        add     bp,4
*        mov     ax,[bp - 4]
*        mov     [bp],ax
*        mov     ax,[bp - 6]
*        mov     [bp - 2],ax
*_?dpl:
]

[ drop               ; d | .        ;スタックトップのデータをひとつ捨てる
*        sub     bp,4
]

[ 2drop              ; d,d | .      ;スタックデータをトップからふたつ捨てる
*        sub     bp,8
]

[ swap               ; d1,d2|d2,d1 ;スタックトップと二番目を交換
*        mov     ax,[bp]
*        xchg    [bp - 4],ax
*        xchg    [bp],ax
*        mov     ax,[bp - 2]
*        xchg    [bp - 6],ax
*        xchg    [bp - 2],ax
]

[ over               ; d1,d2|d1,d2, ;スタックの二番目のデータを複写
*        add     bp,4
*        mov     ax,[bp - 8]
*        mov     [bp],ax
*        mov     ax,[bp - 10]
*        mov     [bp - 2],ax
]

; ==========================================================
;
;                       メモリアクセス用のワード
;
; ==========================================================

[ @                  ; addr | d     ;アドレスからワード値を読み出す
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        mov     ax,es:[bx]
*        mov     [bp],ax
]

[ c@                 ; addr | d     ;アドレスからバイト値を読み出す
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        mov     al,es:[bx]
*        mov     ah,0
*        mov     [bp],ax
]

[ 1+@                ; addr | d     ;アドレスのワード値をインクリメントし読み出す
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        inc     word ptr es:[bx]
*        mov     ax,es:[bx]
*        mov     [bp],ax
]

[ @1+                ; addr | d     ;アドレスのワード値を読み出し後インクリメント
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        mov     ax,es:[bx]
*        inc     word ptr es:[bx]
*        mov     [bp],ax
]

[ !                  ; d , addr | .;ワードデータをアドレスにストア
*        mov     di,[bp]
*        mov     es,[bp - 2]
*        mov     ax,[bp - 4]
*        stosw
*        sub     bp,8
]

[ c!                 ; d , addr | .;バイトデータをアドレスにストア
*        mov     di,[bp]
*        mov     es,[bp - 2]
*        mov     al,[bp - 4]
*        stosb
*        sub     bp,8
]

[ +!                 ; d , addr | .;アドレスのワードデータを増加
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        mov     ax,[bp - 4]
*        add     es:[bx],ax
*        sub     bp,8
]

[ 1+!                ; addr | .     ;アドレスのワードデータをインクリメント
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        inc     word ptr es:[bx]
*        sub     bp,4
]

[ -!                 ; d , addr | .;アドレスのワードデータを減少
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        mov     ax,[bp - 4]
*        sub     es:[bx],ax
*        sub     bp,8
]

[ 1-!                ; addr | .     ;アドレスのワードデータをデクリメント
*        mov     bx,[bp]
*        mov     es,[bp - 2]
*        dec     word ptr es:[bx]
*        sub     bp,4
]

[ full_address       ; off,seg|addr;segをoffの上位ワードに書込む
*        mov     ax,[bp]
*        sub     bp,4
*        mov     [bp - 2],ax
]

[ reg_ds             ; . | seg      ;セグメントレジスタ値を取得
*        add     bp,4
*        mov     [bp],ds
]

[ reg_cs             ; . | seg      ;セグメントレジスタ値を取得
*        add     bp,4
*        mov     [bp],cs
]

[ reg_ss             ; . | seg      ;セグメントレジスタ値を取得
*        add     bp,4
*        mov     [bp],ss
]

[ set_reg_ds         ; d | d        ;セグメントレジスタ値を上位ワードに設定
*        mov     [bp - 2],ds
]

[ set_reg_cs         ; d | d        ;セグメントレジスタ値を上位ワードに設定
*        mov     [bp - 2],cs
```

ワンラインNEWS
ニューテックは，日本電気の「PC-9800」に対応した500MBのハードディスクを発売．インターフェースにバッファを内蔵し，従来の3.8倍の転送能力を実現した．

“crlf”は標準出力に改行を送るワードであり，このワードはスタックを消費せず，またスタックになにも積まない．したがってスタックは空のままである．

これが“_print_count”のスタック遷移から見た動作である．スタック仕様の通りになっていることを確認していただきたい．

次に，これを呼び出すほうのワードを見てみよう．

“main”はまず“$$startup”を呼び出してコアを初期化し，歓迎の挨拶を表示する．ダブルクォートではさまれたものは文字列であり，その先頭アドレスをスタックに積むワードとしてコンパイルされている(オブジェクト生成の解説を参照のこと)．

次に10回のループに入り，“i”によってループの中で現在のループカウンタ(0から始まる)を参照して，それを“print_count”に与えて表示させる．この中で表示までやってしまってももちろんいいが，ワード呼び出しの例として分離したものである．

FORTH系の言語ではデータの受け渡しがスタック経由になっており，数値，演算子，文字列，一般ワードといった区別はあまり意味をもたない．数値や文字列はそれ自身をスタックに積むワードとして認識されるし，演算子にいたってはそもそもそのような区別はなく，単なる一般ワードを演算子と見立てているだけなのである．したがって，FORTH系言語ではプログラムの任意の部分を切り出して独立したワードにすることが簡単にできる．

この特性を使って処理を細かな部品に切り分けてプログラムを構成していくのが，FORTH系言語の上手なプログラミングである．たとえばの話だが，プログラムのあちこちで「3を加える」という処理が行われるとすると，リスト3のようなワードをさっさと定義してしまえばいいのである．ただし，これではワードを独立させる意味があまりない．「3を加える」というのがなにを意味するのかをワード名に反映させてやらなければならない．

さて，リスト2のソースをコンパイルするとリスト4のようなアセンブラのソースができる．“_print_count”は先頭にアンダースコアの付いたワードなので外部から参照されるためのラベルなどは生成されていないことも確認していただきたい．

リスト3

```
[ 3+      ; d | d+3
            3 + ]
```

■FASTプログラミングテクニック

FASTは何度も述べているようにFORTH系の言語であり，ほかの一般的な言語とはスタックの積極的な使用という点に特徴がある．したがってFASTを上手に使うということはスタックを上手に使うということである．以下，スタックの上手な使い方について解説する．

【たくさん積むな】

スタックは先月号でも説明したようにどんどん積み重ねていくような構造であるため，スタックの2番目以降のデータにアクセスするのがめんどうである．スタックのトップと2番目を入れ替える“swap”のようなワードも用意されてはいるが，あまりそれに頼り

リスト4

```
include         core.h

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT , DS:_DATA

f__print_count  proc
;TEST.f(7)      '0' + cprint CRLF ]
        call    f_$$wnumber
        dw      '0'
        call    f__2B_
        call    f_cprint
        call    f_CRLF
        ret
f__print_count  endp

f_main  proc
if      0
extrn   f_main:near
endif
public  f_main
;TEST.f(10)     $$startup
        call    f_$$startup
;TEST.f(11)     "Welcome to FAST" sprint crlf
        call    f_$$cstring
        dw      16
        db      "Welcome to FAST",0
        call    f_sprint
        call    f_crlf
;TEST.f(12)     10 do
        call    f_$$wnumber
        dw      10
        call    f_$$do_entry
$2000:
;TEST.f(13)             i _print_count
        call    f_i
        call    f__print_count
;TEST.f(14)     loop
        call    f_$$loop_entry
        dw      $2000
;TEST.f(15) ]
        ret
f_main  endp

_TEXT   ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'

_DATA   ENDS

        END     f_main

```


ワンラインNEWS
エー・アイ・ソフトは，同社のエキスパートシステム構築用ソフト「大創玄」をサン・マイクロシステムズの「SPARCステーション」に対応させた．

```
]

[ set_reg_ss        ; d | d       ;セグメントレジスタ値を上位ワードに設定
*       mov     [bp - 2], ss
]

; ============================================================
;
;                       文字／文字列を扱うワード
;
; ============================================================

[ c+                ; d, str | .  ;文字を文字列に追加
*       mov     cx, 0ffffh
*       mov     di, [bp]
*       mov     es, [bp - 2]
*       mov     al, 0
*repne  scasb
*       dec     di
*       mov     al, [bp - 4]
*       stosb
*       mov     al, 0
*       stosb
*       sub     bp, 8
]

[ s!                ; str, buf | . ;文字列をバッファにコピー
*       push    ds
*       mov     si, [bp - 4]
*       mov     ds, [bp - 6]
*       mov     di, [bp]
*       mov     es, [bp - 2]
*_sstr1:
*       lodsb
*       stosb
*       cmp     al, 0
*       jnz     _sstr1
*       sub     bp, 8
*       pop     ds
]

[ s+                ; str, buf | . ;文字列をバッファに追加
*       push    ds
*       mov     cx, 0ffffh
*       mov     di, [bp]
*       mov     es, [bp - 2]
*       mov     al, 0
*repne  scasb
*       dec     di
*       push    di
*;
*       mov     cx, 0ffffh
*       mov     di, [bp - 4]
*       mov     si, di
*       mov     es, [bp - 6]
*repne  scasb
*       not     cx
*       push    es
*       pop     ds
*;
*       mov     es, [bp - 2]
*       pop     di
*rep    movsb
*       sub     bp, 8
*       pop     ds
]

[ s=                ; str, str|T/F ;文字列比較
*       push    ds
*       mov     si, [bp]
*       mov     ds, [bp - 2]
*       mov     di, [bp - 4]
*       mov     es, [bp - 6]
*       push    di
*       xor     al, al
*       mov     cx, 0ffffh
*repne  scasb
*       not     cx
*       pop     di
*repe   cmpsb
*       mov     ax, TRUE
*       je      _seq1
*       mov     ax, FALSE
*_seq1:
*       sub     bp, 4
*       mov     [bp], ax
*       pop     ds
]

[ stail             ; str | addr  ;文字列の末尾アドレスを取得
*       mov     di, [bp]
*       mov     es, [bp - 2]
*       xor     al, al
*       mov     cx, 0ffffh
*repne  scasb
*       dec     di
*       mov     [bp], di
]

[ c@stail           ; addr | d    ;文字列の最後の文字を取得
        stail 1- c@
]

[ slen              ; addr | d    ;文字列のバイト数を取得
*       mov     di, [bp]
*       mov     es, [bp - 2]
*       xor     al, al
*       mov     cx, 0ffffh
*repne  scasb
*       not     cx
*       dec     cx
*       mov     [bp], cx
]

[ csearch           ; d, str | adr ;文字列から文字を検索
*       mov     di, [bp]
*       mov     es, [bp - 2]
*       xor     al, al
*       mov     cx, 0ffffh
*       push    di
*repne  scasb
*       not     cx
*       pop     di
*       mov     al, [bp - 4]
*       test    sp, sp
*repne  scasb
*       mov     ax, FALSE
*       jnz     _cs1
*       dec     di
*       mov     ax, di
*_cs1:
*       sub     bp, 4
*       mov     [bp], ax
*       mov     [bp - 2], es
]

[ cupcase           ; d | D       ;小文字を大文字に変換
*       mov     al, [bp]
*       cmp     al, 'a'
*       jb      _cu1
*       cmp     al, 'z'
*       ja      _cu1
*       and     al, 0dfh
*       mov     [bp], al
*_cu1:
]

; ============================================================
;
;                       算術演算などのワード
;
; ============================================================

[ +                 ; 3, 2 | 5    ;加算
*       mov     ax, [bp]
*       sub     bp, 4
*       add     [bp], ax
]

[ 1+                ; 1 | 2       ;インクリメント
*       inc     word ptr [bp]
]

[ 2+                ; 1 | 3       ;2だけ増加
*       inc     word ptr [bp]
*       inc     word ptr [bp]
]

[ -                 ; 3, 2 | 1    ;減算
*       mov     ax, [bp]
*       sub     bp, 4
*       sub     [bp], ax
]

[ 1-                ; 3 | 2       ;デクリメント
*       dec     word ptr [bp]
]

[ 2-                ; 3 | 1       ;2だけ減少
*       dec     word ptr [bp]
*       dec     word ptr [bp]
]

[ *                 ; 6, 2 | 12   ;乗算
*       mov     ax, [bp - 4]
*       mul     word ptr [bp]
*       mov     [bp - 4], ax
*       sub     bp, 4
]

[ /                 ; 6, 2 | 3    ;除算
*       mov     ax, [bp - 4]
*       mov     dx, 0
*       div     word ptr [bp]
*       mov     [bp - 4], ax
*       sub     bp, 4
]

[ mod               ; 7, 2 | 1    ;剰余
*       mov     ax, [bp - 4]
*       mov     dx, 0
*       div     word ptr [bp]
*       mov     [bp - 4], dx
*       sub     bp, 4
]

[ /mod              ; 7, 2 | 1, 3 ;商と剰余
*       mov     ax, [bp - 4]
*       div     word ptr [bp]
*       mov     [bp - 4], dx
*       mov     [bp], ax
]

[ 0<                ; -1 | T      ;0より小さいか?
*       mov     ax, [bp]
*       and     ax, 8000h
*       rcl     ax, 1
*       rcl     ax, 1
*       mov     [bp], ax
]

[ 0>                ; 1 | T/F     ;0より大きいか?
*       mov     ax, [bp]
*       or      ax, ax
*       jz      _zr1
*       and     ax, 8000h
*       rcl     ax, 1
*       cmc
```

過ぎるとかえってプログラムの流れがわかりにくくなる．

とくに，同じくらいの使用頻度のデータがスタック上に複数置かれている場合は，プログラミングが極端にむずかしくなり，できあがったロジックはスタック操作のかたまりになって，肝心のロジック自体の見とおしが悪くなってしまう．

どうしてもそうしなければならない場合は，スタックの順序が入れ替わる場所ごとにその時点のスタックの状態をコメントの形で書いておくことをお勧めする．逆にいうと，そのようなコメントを書かなければならないワードは見直してみる価値があるということでもある．

【順序を考えて積め】

スタック上のデータは，呼び出すワードへの引数になることが多いので，呼び出す段になって順序を入れ替えたり2番目をもってきたりということがないように，データが自然に流れるようにしてやらなければならない．順序を考えてデータを積むということは，無理のない処理の流れを作ること，そしてそれを受け入れるように下位のワードを設計することでもある．

ちょっとしたコツをいうと，ワードに渡すデータは「使用頻度の低いもの」→「使用頻度の高いもの」→「すぐに消費されるもの」という順に積むのが上手な方法である．すぐに消費されるものがスタックトップにくるような積み方だ．こうすると，スタックトップはすぐに消費されてなくなり，次に使用頻度の高いデータが現われる．

ワードの再利用や汎用性という考え方からは，影響を与えるもの，与えられるものがトップにくるような構成が望ましい．たとえば，決まったアドレスから指定されたバッファにデータを切り出してくるようなワードは，スタックトップにバッファが置かれるような構成にしておくと，「Aに切り出してくるワード」「Bに切り出してくるワード」というのが作りやすくなる．反対に指定されたアドレスから決まったバッファにデータを切り出してくるようなワードは，スタックトップに切り出しもとのアドレスを置くほうが上位のワードの構成が楽になることは容易に想像がつくだろう．

なかなかこうきれいにはいかないものだが，一応の指針としておぼえておくといい．

【退避場所を有効に使え】

どうしてもスタックに同じ程度の使用頻度のデータが複数のってしまうことがある．こういう場合はリターンスタックを使うというのも1つの手だ．リターンスタックは文字どおり上位ワードへの復帰のためのもので，通常は遊んでいる．ここに1つデータを移すことでパラメータスタックが格段に楽になる．リターンスタックに関してはパラメータスタックから移し，または戻し，参照するといった基本的なワードが用意されているので，有効に利用したほうがいい．基本ワードの構成から見ると，参照のみで変更されないデータをリターンスタックに移動するというのが上手な使い方だが，自分でリターンスタック上のデータを操作するワードを定義するならばそれにこだわる必要はない．

ところで，リターンスタック上のデータを操作するワードを作る場合には，そのワードが呼び出されたときにはすでにそのワードからの戻り番地がCPUによって積まれていることを忘れてはならない．

例をあげると，“ i ”というワードはリターンスタックトップのデータを参照しパラメータスタックに積むものという説明がされているが，厳密にはこのワードが呼び出された時点のCPUスタックポインタから1ワード分（つまりこのワードを呼び出したワードへの復帰アドレスの分だけ）下がったところの2ワードを参照するものである．したがって自分がリターンスタックトップに置いたデータの1／3を取り出すワードのつもりで，リスト5のようなワードを定義してこれを呼び出しても思ったような動作はしない．自分がリターンスタックに積んだデータは自分が“ i ”で参照しなければならないのである．

もう1つ，リターンスタックにデータを移した場合は，ワードの終了前に必ずそれを捨てるかパラメータスタックに戻しておかなければならない．リターンスタックはCPUスタックであり，ワードの戻り番地を積むものであることから，理由は明白であろう．

【変数の使用を嫌うな】

FORTHだからといって，なにがなんでもスタックを使わなければならないというものではない．マルチタスクなどの必要性から再入可能なワードにするのならば別だが，そうでなければ，処理の都合でスタックが重くなるようならば変数を使ったほうが楽にプログラムが書ける．

ただし，FORTH系の言語ではメモリアクセスはスタック操作ほどには効率がよくないので，それを心得て使うことをお勧めする．FORTH系の言語にとって変数とはそれが置かれているアドレスであり，それをさらに「読む」あるいは「書く」というワードと組み合わせて使用しなければならないから

ワンラインNEWS
日本電気は，コンパクトなパーソナルワープロ「文豪ハイパー7」を発売した．同社でデスクトップ型に液晶画面を初採用した．奥行きを従来の2/3程度と薄くした．

```
*       rcl     ax,1
*_zrl:
*       mov     [bp],ax
]

[ 0=                    ; 0 | T/F      ;0であるか?
*       mov     ax,0
*       sub     ax,[bp]
*       mov     ax,0
*       cmc
*       rcl     ax,1
*       mov     [bp],ax
]

[ 0<>                   ; 0 | T/F      ;0でないか?
*       mov     ax,0
*       sub     ax,[bp]
*       mov     ax,0
*       rcl     ax,1
*       mov     [bp],ax
]

[ =                     ; 2,2 | (TRUE);等しいか?
*       mov     ax,[bp]
*       cmp     ax,[bp - 4]
*       mov     ax,FALSE
*       jnz     _eql
*       mov     ax,TRUE
*_eql:
*       mov     [bp - 4],ax
*       sub     bp,4
]

[ <>                    ; 2,2 | F      ;等しくないか?
*       mov     ax,[bp]
*       cmp     ax,[bp - 4]
*       mov     ax,FALSE
*       jz      _eql
*       mov     ax,TRUE
*_neql:
*       sub     bp,4
*       mov     [bp],ax
]

[ >                     ; 3,2 | (TRUE);大きいか?
*       mov     ax,[bp]
*       cmp     [bp - 4],ax
*       mov     ax,TRUE
*       ja      _bgl
*       mov     ax,FALSE
*_bgl:
*       sub     bp,4
*       mov     [bp],ax
]

[ <                     ; 3,2 | F      ;小さいか?
*       mov     ax,[bp]
*       cmp     [bp - 4],ax
*       mov     ax,TRUE
*       jb      _sml
*       mov     ax,FALSE
*_sml:
*       sub     bp,4
*       mov     [bp],ax
]

[ >>>                   ; 4 | 8        ;1ビット右シフト
*       shr     word ptr [bp],1
]

[ >>                    ; 4,2 | 16     ;右シフト
*       mov     cl,[bp]
*       sub     bp,4
*       mov     ax,[bp]
*       shr     ax,cl
*       mov     [bp],ax
]

[ <<<                   ; 4 | 2        ;1ビット左シフト
*       shl     word ptr [bp],1
]

[ <<                    ; 16,2 | 4     ;左シフト
*       mov     cl,[bp]
*       sub     bp,4
*       mov     ax,[bp]
*       shl     ax,cl
*       mov     [bp],ax
]

[ &                     ; d,d | d      ;ビット演算AND
*       mov     ax,[bp]
*       sub     bp,4
*       and     [bp],ax
]

[ |                     ; d,d | d      ;ビット演算OR
*       mov     ax,[bp]
*       sub     bp,4
*       or      [bp],ax
]

[ &&                    ;T/F,T/F | T/F;論理演算AND
*       mov     bx,FALSE
*       mov     ax,[bp]
*       sub     bp,4
*       or      ax,ax
*       jz      _adl
*       mov     ax,[bp]
*       or      ax,ax
*       jz      _adl
*       mov     bx,TRUE
*_adl:
*       mov     [bp],bx
]

[ ||                    ;T/F,T/F | T/F;論理演算OR
*       mov     bx,FALSE
*       mov     ax,[bp]
*       sub     bp,4
*       or      [bp],ax
*       jz      _orl
*       mov     bx,TRUE
*_orl:
*       mov     [bp],bx
]

[ <range>               ; d,L,H | T/F ;範囲内か?
*       mov     bx,FALSE
*       mov     ax,[bp - 8]
*       cmp     ax,[bp - 4]
*       jb      _irgl
*       cmp     ax,[bp]
*       ja      _irgl
*       mov     bx,TRUE
*_irgl:
*       mov     [bp - 8],bx
*       sub     bp,8
]

[ >range<               ; d,L,H | T/F ;範囲外か?
        <range> 0 =
]

[ cwd                   ; d | d       ;ダブルワードに変換
*       mov     ax,[bp]
*       cwd
*       mov     [bp - 2],dx
]

; =====================================================
;
;                       入出力関連のワード
;
; =====================================================

[ sprint_handle         ; addr,h | .  ;ハンドル指定の文字列表示
*       push    ds
*       mov     di,[bp - 4]
*       mov     ax,[bp - 6]
*       mov     ds,ax
*       mov     es,ax
*       mov     cx,0ffffh
*       mov     dx,di
*       mov     al,0
*repne  scasb
*       not     cx
*       dec     cx
*       mov     bx,[bp]
*       mov     ah,40h
*       int     21h
*       pop     ds
*       sub     bp,8
]

[ sprint                ; addr | .    ;文字列表示
        (STDOUT) sprint_handle
]

[ cprint_handle         ; d , h | .   ;ハンドル指定の文字表示
*       push    ds
*       mov     bx,[bp]
*       lea     dx,[bp - 4]
*       mov     cx,1
*       mov     ax,ss
*       mov     ds,ax
*       mov     ah,40h
*       int     21h
*       sub     bp,8
*       pop     ds
]

[ cprint                ; d | .       ;文字表示
        (STDOUT) cprint_handle
]

[ crlf_handle           ; handle | .  ;ハンドル指定の改行
        0dh over cprint_handle
        0ah swap cprint_handle
]

[ crlf                  ; . | .       ;改行
        (STDOUT) crlf_handle
]

[ key_wait              ; . | .       ;キー入力待ち
*       mov     ah,08h
*       int     21h
*       add     bp,4
*       mov     ah,0
*       mov     [bp],ax
]

[ cinput                ; . | d       ;エコーなしキー入力
*       mov     ah,7
*       int     21h
*       mov     ah,0
*       add     bp,4
*       mov     [bp],ax
]

[ cinpute               ; . | d       ;エコーつきキー入力
        cinput dup cprint
]

[ fcinput               ; h | d       ;ハンドル指定の文字入力
*       push    ds
*       mov     bx,[bp]
```

である．

コンパイル・リンクのしかた

まず最初にFASTがCORE. FをコンパイルしてできたマMASMのソースファイルをアセンブルしてCORE. OBJを作成する．次に先月作成したFAST. OBJとCORE. OBJをリンクしてFASTコンパイラを完成させる．

編集部注：紙面の都合でMASMのソースファイルは掲載しておりません．付録のディスクには収録されておりますので，そちらをご利用ください．

```
A> masm CORE;
A> link FAST CORE;
```

次に，このコンパイラを使ってプログラムを書いたり，自分自身の機能改良をする場合だが，以下のようなバッチファイル“COMPILE. BAT”を作っておくと便利である（リスト6）．

こうしておくと，

```
A> compile program
```

とするだけで，コンパイル，ヘッダファイルの作成，アセンブルまでを自動的に実行してくれる．プログラマーズ・ページの読者ならば，このバッチの意味するところは理解できるだろう．ヘッダファイルの作成に“find”を使っているのでソース中に“extrn”が入っている行も抽出してしまう．いずれ先頭が“extrn”で始まっている行だけを抽出するツールをFASTで書くか，FASTコンパイラ自体にヘッダファイルを作成する機能を追加しようと思う．

リスト5

```
[ R/3
        i 3 / ]
```

リスト6

```
fast %1.f >%1.asm
find <%1.asm "extrn">%1.h
masm %1;
```

```
*       mov     ax, ss
*       mov     ds, ax
*       mov     dx, bp
*       mov     cx, 1
*       mov     ah, 3fh
*       int     21h
*       jc      _fci1
*       cmp     ax, 1
*       jz      _fci2
*_fci1:
*       mov     word ptr [bp], EOF
*_fci2:
*       pop     ds
]

[ fopen                 ; fn, m|h, T/F  ;ファイルオープン
*       push    ds
*       mov     al, [bp]
*       mov     dx, [bp - 4]
*       mov     ds, [bp - 6]
*       mov     ah, 3dh
*       int     21h
*       mov     [bp - 4], ax
*       mov     bx, TRUE
*       jnc     _fo1
*       mov     bx, FALSE
*_fo1:
*       mov     [bp], bx
*       pop     ds
]

[ fclose                ; h | T/F       ;ファイルクローズ
*       mov     ah, 3eh
*       mov     bx, [bp]
*       int     21h
*       mov     ax, TRUE
*       jnc     _fc1
*       mov     ax, FALSE
*_fc1:
*       mov     [bp], ax
]

[ fread                 ; size, buf, h | size, T/F
*       push    ds
*       mov     dx, [bp - 4]
*       mov     ds, [bp - 6]
*       mov     cx, [bp - 8]
*       mov     bx, [bp]
*       sub     bp, 4
*       mov     ah, 3fh
*       int     21h
*       mov     [bp - 4], ax
*       mov     ax, TRUE
*       jnc     _fr1
*       mov     ax, FALSE
*_fr1:
*       mov     [bp], ax
*       pop     ds
]

[ fsinput               ; buf , h|T/F;ハンドル指定の一行入力
;
;注意：このバージョンでは複数のファイルを扱えない。
;
        #rfh !
        #roff !
        0 #roff @ c!
        #rboff @ 0<>
        if      (CR) #rboff @ csearch ?dup 0<>
                if      0 over c! #rboff @ #roff @ s!
                        1+ dup #rboff !
                        c@ (LF) =
                                if      #rboff 1+!
                                then
                        (TRUE) return
                else    #rboff @ #roff @ s!
                then
        then
        (RBUFSIZ-2) #rbuf #rfh @ fread 0=
                if      drop (FALSE) return
                then
        dup #rsiz ! 0=
                if      (FALSE) return
                then
        0 #rbuf #rsiz @ + c!
        #rbuf dup c@ (LF) =
                if      1+
                then
        #rboff !
        (CR) #rboff @ csearch ?dup 0=
                if      #rboff @ #roff @ s+
                        0 #rboff !
                         (TRUE) return
                then
        0 over c!
        #rboff @ #roff @ s+
        1+ dup #rboff !
        c@ (LF) =
                if      #rboff 1+!
                then
        (TRUE)
]

[ sinput                ; addr | T/F  ;一行入力
        0 fsinput
]

_TEXT   ENDS

; ==========================================================
;
;                       データ定義
;
; ==========================================================

_DATA   SEGMENT WORD PUBLIC 'DATA'

rfh     dw      (?)
rboff   dw      (?)
rsiz    dw      (?)
roff    dw      (?)
rbuf    db      RBUFSIZ dup (?)

pspbuf  dw      (?)

newpsp  db      256 dup (?)

_DATA   ENDS

_STACK  SEGMENT STACK
        dw      1024 dup (?)
_STACK  ENDS

        END
```


ワンラインNEWS
YHPは，米HP社独自のRISCチップを採用した「HP9000シリーズ800UNIXサーバーファミリ」の追加機種6モデルを発売．現行機種に比べ価格性能比を2倍以上に向上．

筆者平謝りの図

6月号の「MS-DOSツール・ラボ」のプログラムは式の評価を正しく行いません．というのは，まず記事中の構文図がまちがっているため，それを元にコーディングしたのでは，全くおかしな結果となってしまうのです．構文図の単純式と項は下のようにするべきです．記事のままでは，たとえば3＋3＋3すら正しくない結果を生みます．また，プログラムで，単項の‘−’の処理は完全なバグで，本当は単純式ではなく項の評価結果を負にしなければなりません．
（飯田浩二さんからのお便り；図は省略しました．）

まことに申しわけない．連載6月号の式の評価にとんでもないバグがあった．くどくどいいわけはしないが，とにかくまちがいである．なにをどう勘違いしたものか，構文図のいちばん大事な部分がちょうど鏡に写したように左右逆になっている．思い込みとは恐ろしいもので，校正段階でも筆者の2つの節穴を通り過ぎてしまった．構文図をまちがって書いたので，それをそのとおりに展開したプログラムがまちがっていることはいうまでもない（もちろん威張れた話しではない）．

筆者と違って専門教育を受けておられるらしい読者の穏やかなご指摘とともに，編集者の「あのですね……」という冷たいお言葉を頂戴して全身から血の気がひいた．うまい具合に事務所がビルの8階にあるので，「いっそ死んでおわびを」をやるのには都合のいい条件が整っているのだが，飛び降りる前にバグの修正だけはしておこうと思った（当然である）．

「もうしませんから許して」が通用するのは5才までと相場が決まっているので，この年になるとなんともいいようがない．ともあれプログラムの修正をしたので，編集部に無理をお願いしてディスクサービスをしていただくことにする．

修正版の構文図を見ていただければ一目瞭然であるが，ループにするべきものをしていなかったのであるから，この部分の修正は簡単である．RETで返っていたものをJMPで頭に戻るようにすればいい．

おわびの印にもなんにもならないが，単項の‘−’の評価を因子の部分で行う方法をご提案する．こうすると「−−3」というのも「3」として評価できる（じつはこの単項の評価にもバグがあった）．やり方は非常に簡単で，‘−’を見つけると自分自身（因子）を再帰的に呼び出し，結果を負にするのである．

どうもご迷惑をおかけして申しわけない．まことにまことに申しわけない．ひたすら平謝りの筆者である．

（編集部：修正版のプログラムは，本誌付録ディスクに収録しました．）

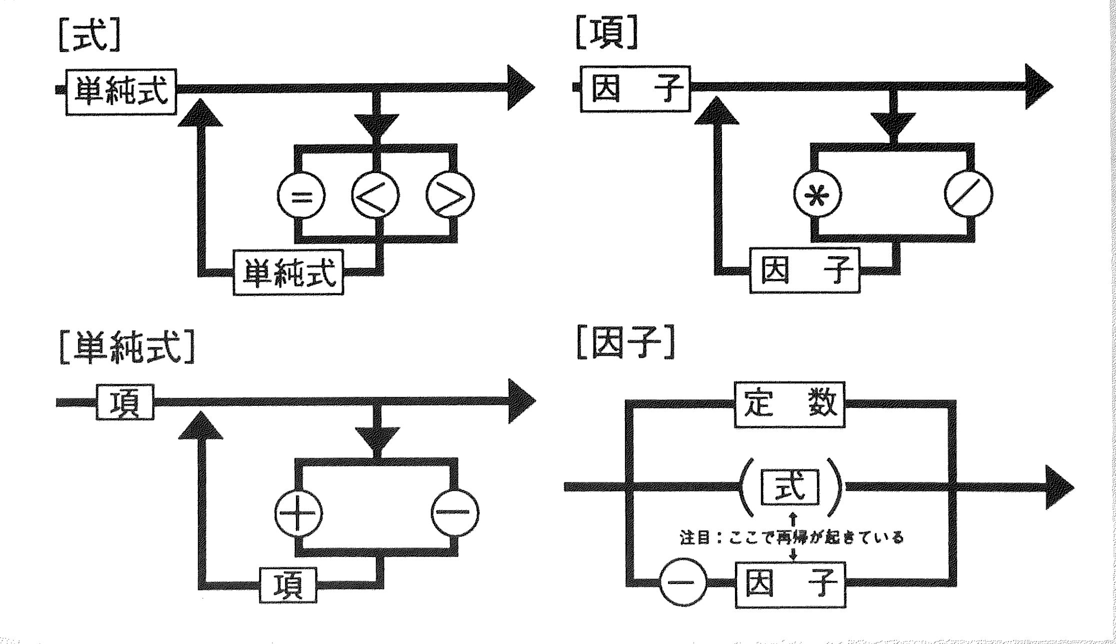
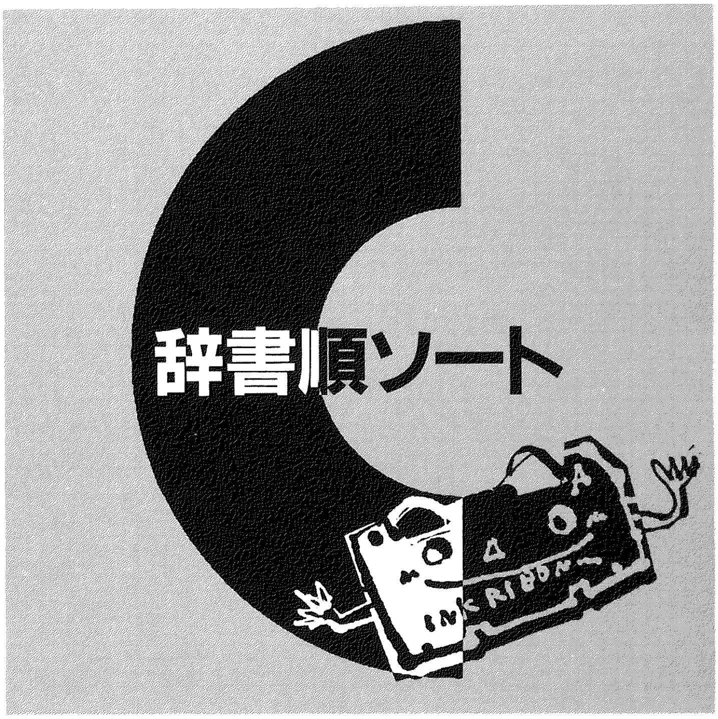


Illustration:
Masami Nozue

実践C塾

基礎テクニック篇　第4回



二代目塾頭　大野元久

文字列のソートを行う場合，ソートのアルゴリズム以前にどのような比較規則を用いてソートするかという問題がある．英字のみの文字列ならば簡単なようにも思えるが，案外複雑である．日本語文字列のソートにおける比較規則の複雑さはいうまでもない．今回は，この文字列ソートの問題を取り上げた．

文字列の比較

今回は，文字列の比較を取り上げます．たかだか文字列の比較と馬鹿にしてはいけません．文字列をソートするために気の利いた比較関数を用意するには，Cのライブラリ関数では不十分なことが多いのです．まず，その例を示してみましょう．

名簿管理のようなデータベースプログラムを作るとします．せっかくのデータベースプログラムですから，データを参照しやすいように，名前順でソートできるようにしたいですね．データをソートする場合は，データの型に応じて任意の2つのデータの大小関係を決めてやらなければなりません．

さて，この大小関係はどのように決めればよいでしょうか．説明を簡単にするため，まず英字について考えます．この場合，文字列の大小関係を決めるための比較関数としてstrcmpやstricmpなどが思い浮かびます．

しかし，strcmpは文字列をASCIIコード順でしか比較しないため，文字列の一致を判別する目的以外で使うには不十分なことがあります．たとえば，strcmpではすべての英大文字は，すべての英小文字より小さいものとみなされますので，America，abort，autoという文字列があれば，America<abort<autoという結果になってしまいます．「アルファベット順」にソートしたいのであれば，abort<America<autoという結果になるべきでしょう．

リスト1 (estrcmp.c)

```
#include <string.h>

/*
 *  文字列を英大・小文字を区別せずに比較し、一致する場合だけ
 *  英大文字の方が小さいものとみなす。
 *
 *  戻り値  負  s1 < s2
 *           0  s1 == s2
 *          正  s1 > s2
 */
int estrcmp(const char *s1, const char *s2)
{
    int ret;

    ret = stricmp(s1, s2);
    if (ret != 0) return ret;
    return strcmp(s1, s2);
}

```

それなら，英字の大・小文字を無視するstricmpを使ったらどうでしょうか．stricmpは，大･小文字をまったく区別しないため，Stringとstringのように文字の大・小だけが違う文字列では，どちらが先にくるのかわかりません．文字が一致する場合は，小文字を優先するということができないわけです．

では，リスト1 (estrcmp. c)のように，これらを組み合わせればどうでしょうか．これも十分ではありません．U. S. A. という文字列とUnitedという文字列を比較してみましょう．‘.’の文字コード(0x2E)がnの文字コード(0x6E)よりも小さいことを知っているプログラマなら，U. S. A. がUnitedよりも小さいものと評価されて当然だと思うかもしれません．しかし，一般的には文字列中の‘.’は無視して比較し，U. S. A. はUnitedよりも後ろにくるべきです．

日本語の文字列を比較する場合は，さらに複雑な比較規則が必要となります．同じ文字についてでさえ，半角カタカナ，全角ひらがな，全角カタカナという異なった表記がされます．文字コードについても，濁点，半濁音，促音，拗音，長音など，さまざまな条件を考慮しなければならないのです．Turbo C++やMicrosoft Cのライブラリ関数にあるjstrcmpは，半角カナ文字が漢字コードに挟まれないように順序を変更しているだけで，こうした文字種に関する配慮はされていません．

名簿管理の例でも，「よみかた」を全角ひらがなだけに統一するという制約をつければある程度問題はなくなりますが，辞書順にソートすることができるわけではありませんし，なによりも使いやすいアプリケーションを作成するためには，こうした制約はなくしたいものです．

ソートとデータ型

どんなデータ型でも，任意の2つの要素の大小関係を決めることができれば，そのデータ型の配列をソートすることができます．Cでは，比較関数を用意すればqsort関数がこのソート処理を実行してくれます．qsortの引数

は，リスト２のとおりです．

ここで，baseはソートされる配列の先頭へのポインタ，numはソートされる配列の個数，widthは１つの要素のバイト数，fcmpは比較関数へのポインタです．

比較関数は，対象となるデータ型へのポインタを２つ引数としてとり，第１引数の指す内容から第２引数の指す内容を引いた値を返すものと考えます．実際には比較した結果，２つの引数が一致すれば０を，第１引数が第２引数より大きければ正の値を，第１引数が第２引数より小さければ負の値を返すということが重要です．このような比較関数を用意すれば，qsortはデータを昇順に（小さいほうから大きいほうへ）ソートします．データを降順に（大きいほうから小さいほうへ）ソートするときは，比較関数が返す値の正負を逆にします．

単純な整数型のデータ配列をソートするならば，リスト３（qtest. c）のようにすればよいでしょう．比較関数の引数にconst修飾子をつけているのは，比較関数は与えられた引数が指す内容を変更しないということをあらわすためです．Turbo C++（Borland C++）のユーザーの方は，qsort関数の最後の引数につけられている（int (*) (const void *, const void *)）というキャストに注意してください．Turbo C++では，引数の型が必要以上に厳密に判断されてしまうためか，比較関数の前には常にこのキャストが必要となります．

比較関数が返すのはint型ですから，int以外の型では引算の結果を正しくint型に反映させなければなりません．Turbo C++やMicrosoft Cのように，int<longとなっている処理系では，long型のデータ配列をソートするためにリスト４のような関数を使ってはいけません．

long型の引算の結果がintの範囲に収まらないと，超過した分が切り捨てられてしまい正しい結果を得られなくなるからです．200000と100000を比較すると100000という正の値になりますが，これをint型に変換すると-31072となり負の値になります．つまり200000が100000よりも小さいものと評価されてしまうわけで，このような関数では正しいソートは行われません．

この場合は，リスト５のような関数を用意します．

qsortに渡すデータ配列については，次のように一般化できます．データ配列が次のような形式であれば，

データ型　配列名［］;

qsortには，リスト６のような引数を渡します．

第２引数に与えられている“sizeof 配列名/sizeof 配列名［0］”というのは，配列の要素数を得るためによく使われる手法です．sizeof 配列名が配列

リスト２

```
void qsort(void *base, size_t num, size_t width,
           int (*fcmp)(const void *, const void *));
```

リスト３（qtest.c）

```
/*
 *  stest.c: qsort を使った整数データのソート
 */
#include <stdio.h>
#include <stdlib.h>

int table[] = { 335, 33, 217, 536, 195 };

#define TBLMAX (sizeof table / sizeof table[0])

int compare(const int *p1, const int *p2)
{
    return *p1 - *p2;
}

void dsptable(void)
{
    int i;

    printf("-- table list --¥n");
    for (i = 0; i < TBLMAX; i++)
        printf("table[%d] = %d¥n", i, table[i]);
}

void main()
{
    dsptable();
    qsort(table, TBLMAX, sizeof sizeof table[0],
        (int (*)(const void *, const void *))compare);
    dsptable();
}
```

リスト４

```
int longcmp(const long *p1, const long *p2)
{
    return (int)(*p1 - *p2);
}
```

リスト５

```
int longcmp(const long *p1, const long *p2)
{
    if (*p1 > *p2) return 1;
    if (*p1 < *p2) return -1;
    return 0;
}
```

リスト６

```
qsort(配列名, sizeof 配列名/sizeof 配列名[0], sizeof 配列名[0],
    (int (*)(const void *, const void *))比較関数名);
```

リスト７

```
#define TBLMAX (sizeof 配列名 / sizeof 配列名[0])
```


ワンラインNEWS
日本DECは，IBMのPS/55シリーズに対応したパソコン統合ソフト「PATHWORKS」を開発した．同ソフトはすでに日本電気のPC-9801シリーズなどに対応している．

全体のバイト数を，sizeof 配列名[0]が配列の第1要素(つまり要素1つの)のバイト数をあらわします．配列の要素数は，ほかでも使い道がありますから，リスト7のように定義しておくとよいでしょう．

比較関数の引数には，注意を払わなければなりません．比較関数に渡される引数は，配列の要素へのポインタ型になります．配列が文字配列の配列(文字の2次元配列）であらわされる場合と，文字列へのポインタの配列であらわされる場合とで比較してみましょう．

文字列の2次元配列で定義した例をリスト8に示します．

ここで，配列の要素の型はchar[20]，つまり文字配列そのものです．したがって，引数関数はリスト9のような引数をとることになります．

しかし，「20個の文字要素をもつ配列型」を比較する関数はありませんし，s1は文字配列の先頭を指しているわけですから，もっと簡単にリスト10のように比較関数を定義できます．

実際には，estrcmpとstrcmpは同じものですから，qsortにはstrcmpを渡すことができます．この配列は，リスト11のようにしてソートできます．

次に，文字列をポインタの配列で定義した例をリスト12に示します．

配列の要素の型はchar＊ですから，比較関数に渡される引数の型は，char＊へのポインタ型，つまりchar＊＊となります（リスト13).

xstrcmpとstrcmpは，引数の型が違いますから置き換えることはできません．この配列は，リスト14のようにしてソートできます．

これまでは，単純なデータをソートしただけですが，一般にデータをソートする場合は，それに付随するデータが必要となるものです．たとえば，人名をソートしてもそれに付随する住所や電話番号などのデータがいっしょに入れ換えられなければ何にもなりません．こうしたデータをリスト15のような構造体にまとめるとします．

前述の例と同じく，配列の要素を構造体そのものと構造体へのポインタであらわしてデータをソートする例を，それぞれリスト16（qtest2.c）とリスト17（qtest3.c）に示します．

qsortは，配列を要素単位で入れ換えます．前述の文字の2次元配列の例では，配列の要素が20バイトあるため，比較結果によって20バイト単位で入れ換えが発生します．しかし，ポインタの配列の例のように配列の要素をポインタ型にしておけば，この入れ換えはポインタの大きさ，つまりせいぜい2バイトか4バイトですみます．qsortは，効率のよいアルゴリズムを使っていますが，データ要素数が増えればそれだけ入れ換えが発生する数も増えます．内部で再帰処理を行っていますので，データ要素が大きいとスタックの消費量が増え無用なバグのもとになりかねません．高速で安全なソート

リスト8

```
char table[5][20] = {
    "Shimizu Takashi",
    "Kahei Suzuki",
    "Sugino Kimiko",
    "Ueda Sumire",
    "Yosiyama Mituhiro",
};
```

リスト9

```
int estrcmp(const char (*s1)[20], const char (*s2)[20]);
```

リスト10

```
int estrcmp(const char *s1, const char *s2)
{
    return strcmp(s1, s2);
}
```

リスト11

```
qsort(table, sizeof table/sizeof table[0], sizeof table[0],
        (int (*)(const void *, const void *))strcmp);
```

リスト12

```
char *table[5] = {
    "Shimizu Takashi",
    "Kahei Suzuki",
    "Sugino Kimiko",
    "Ueda Sumire",
    "Yosiyama Mituhiro",
};
```

リスト13

```
int xstrcmp(const char **s1, const char **s2)
{
    return strcmp(*s1, *s2);
}
```

リスト14

```
qsort(table, sizeof table/sizeof table[0], sizeof table[0],
        (int (*)(const void *, const void *))xstrcmp);
```

リスト15

```
struct person {
    char name[NAME_MAX];        /* 名前 */
    char addr[ADDR_MAX];        /* 住所 */
    char tel[TEL_MAX];          /* 電話番号 */
};
```

処理のためには，できるだけソートの対象をポインタの配列で行うとよいでしょう．

英字の辞書順ソート

英字をソートするときの基本は，いうまでもなくアルファベット順にソートすることです．しかし，それがすべてではありません．英和辞典を見てみましょう．アルファベット順に並んではいますが，綴りが同じ場合は大文字のある単語は必ず小文字の単語の後ろにきています．また，ハイフン（—）やドット（.）などがついているもののほうがついていないものよりも後ろにきています．

文字列を「辞書順」にソートしようというときは，「同じ読みの場合に何を優先するか」ということが大切になります．このため，同じ文字列を簡単な規則からより複雑な規則へ，何度も比較しなおす必要があります．このとき，何を優先するかという規則をきっちり決めておかなければなりません．

今回は，研究社の『リーダーズ英和辞典』を参考にして次のような規則によって比較することにしました．

英字の比較規則

①最初は英数字のみに注目して比較する．英字の大・小文字の区別はしない．

②①では文字列が一致してしまう場合，記号を含めて比較する．ある位置で片方の文字列にのみ記号があらわれた場合は，記号のないものよりも大きい(つまり後ろ)ものとする．ここで，スペース，ドット(省略記号として使われる)，ハイフン（文字の区切りとして使われる）についてはスペース類文字として比較の対象としない．

③②では文字列が一致してしまう場合，英大・小文字を区別して比較する．ある位置で英字の大・小文字が異なる場合は，大文字のあらわれた単語を小文字のあらわれた単語よりも大きいものとする．

④③では文字列が一致してしまう場合，スペース類文字を含めて比較する．ある位置で片方の文字列にのみスペース類文字があらわれた場合は，スペース類文字のないものよりも大きい(つまり後ろ)ものとする．

⑤以上で2つの文字列に違いがなければ，両者を一致するものとみなす．

一般に，英和辞典では単語の語尾が変形しただけのものは並べて記載する

リスト16　(qtest2.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NAME_MAX 20
#define ADDR_MAX 40
#define TEL_MAX 20

struct person {
    char name[NAME_MAX];    /* 名前 */
    char addr[ADDR_MAX];    /* 住所 */
    char tel[TEL_MAX];      /* 電話番号 */
};

int personcmp(struct person *p1, struct person *p2)
{
    return strcmp(p1->name, p2->name);
}

struct person table[] = {
    { "Shoeisha", "Chiyodaku", "03-XXXX-3333" },
    { "Borland Japan", "Minatoku", "03-XXXX-1111" },
    { "Microsoft", "Sinjuku", "03-XXXX-2222" },
};

#define TBLMAX (sizeof table / sizeof table[0])

void dsptable(void)
{
    int i;

    printf("-- table list --¥n");
    for (i = 0; i < TBLMAX; i++)
        printf("table[%d] = %s, %s, %s¥n",
            i, table[i].name, table[i].addr, table[i].tel);
}

void main()
{
    dsptable();
    qsort(table, TBLMAX, sizeof table[0],
        (int (*)(const void *, const void *))personcmp);
    dsptable();
}
```

リスト17　(qtest3.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NAME_MAX 20
#define ADDR_MAX 40
#define TEL_MAX 20

struct person {
    char name[NAME_MAX];    /* 名前 */
    char addr[ADDR_MAX];    /* 住所 */
    char tel[TEL_MAX];      /* 電話番号 */
};

int personcmp(struct person **p1, struct person **p2)
{
    return strcmp((*p1)->name, (*p2)->name);
}

struct person shoueisha = { "Shoeisha", "Chiyodaku", "03-XXXX-3333" };
struct person borland = { "Borland Japan", "Minatoku", "03-XXXX-1111" };
struct person microsoft = { "Microsoft", "Sinjuku", "03-XXXX-2222" };

struct person *table[] = {
    &shoueisha, &borland, &microsoft,
};

#define TBLMAX (sizeof table / sizeof table[0])

void dsptable(void)
{
    int i;

    printf("-- table list --¥n");
    for (i = 0; i < TBLMAX; i++)
        printf("table[%d] = %s, %s, %s¥n",
            i, table[i]->name, table[i]->addr, table[i]->tel);
}

void main()
{
    dsptable();
    qsort(table, TBLMAX, sizeof table[0],
        (int (*)(const void *, const void *))personcmp);
    dsptable();
}
```


ワンラインNEWS
ランドコンピュータは，企業内教育のSI事業に進出する．すでに出荷されているCAI用ソフトを軸に，コンサルティングからSIまでを請け負う．

など，規則よりも意味を優先することがありますが，そのような「不規則さ」はこのプログラムでは実現できません．

この規則でプログラミングした例がリスト18 (ecomp. c) です．使用例をリスト19 (ectest. c) に示します．

リスト19の実行結果は図1のようになります．ここでは，ソート前とソート後の配列tableの内容が表示されていますが，前述の規則にしたがって正しくソートされていることを確認してください．“sand”は“San Francisco” よりも前に，“S. O. S” は “South Africa” よりも前にソートされています．

プログラムの解説

先ほど述べた「英字の比較規則」にしたがってecomp関数を作成します．

ecomp関数は，文字列がまったく同じ場合にはただちに0を返すことができるよう，先頭でstrcmpで一致するかどうかだけを確認し，一致しない場合は，比較規則の番号に対応して文字コードを返すechar関数を呼び出して，2つの文字列の先頭から順に比較処理を行います．

echar関数は，比較規則で無視すべき文字を返しません．したがって，ecomp関数は返される文字コードだけを比較すればよいわけです．ただし，段階が進むことで有効になった文字あるいは区別の対象となる文字については，大きいと評価したい(後ろにしたい)文字コードの上位バイトを1にして，そうでない文字より必ず後ろにくるようにします．たとえば，foobarとfoo+barは段階①では区別されませんが，段階②では区別されます．このとき，echar関数が返す4文字目はそれぞれ‘b’と‘+’となりますが，この

リスト18 (ecomp.c)

```
/*
 *  ecomp.c: 英字文字列の比較関数
 *      written by Motohisa Ohno.
 */
#include <stddef.h>
#include <string.h>
#include <ctype.h>

enum echar_phase_t {
    ECHAR_PHASE_1,      /* 英数字（大小文字を無視） */
    ECHAR_PHASE_2,      /* 英数字、記号（スペース類以外） */
    ECHAR_PHASE_3,      /* 英数字（大小文字を区別）、記号 */
    ECHAR_PHASE_4,      /* 英数字、記号（スペース類を含む） */
};

/*
 *  スペース類とみなす記号を判別するマクロ
 */
#define isspacex(x) (isspace(x) || (x) == '-' || (x) == '_' || (x) == '.')

/*
 *  文字コード取得ルーチン
 */
static const char *echar(int type, const char *str, unsigned *ret)
{
    if (str == NULL) {
        *ret = 0;
        return NULL;
    }

    switch (type) {
    case ECHAR_PHASE_1:
        /* 英字を大小文字の区別なし（ここでは大文字に変換する）で返す */
        /* これにより、アルファベット順に並べられる */
        while (*str && !isalnum((unsigned char)*str))
            str++;
        *ret = toupper((unsigned char)*str);
        return *str ? str+1 : NULL;

    case ECHAR_PHASE_2:
        /* 上記に加え、スペース類以外の記号を返す */
        /* 記号のないものの方が優先される */
        while (*str && isspacex((unsigned char)*str))
            str++;
        *ret = toupper((unsigned char)*str);
        if (*str && !isalnum((unsigned char)*str))
            *ret += 0x100;              /* 記号には重みをつける */
        return *str ? str+1 : NULL;

    case ECHAR_PHASE_3:
        /* 上記に加え、英字の大小文字が区別される */
        /* 英小文字の方が優先される */
        while (*str && isspacex((unsigned char)*str))
            str++;
        *ret = toupper((unsigned char)*str);
        if (isupper((unsigned char)*str))
            *ret += 0x100;              /* 大文字につける重み */
        return *str ? str+1 : NULL;

    case ECHAR_PHASE_4:
        /* 上記に加え、スペース類文字を返す */
        /* スペース類文字のないものの方が優先される */
        *ret = toupper((unsigned char)*str);
        if (isspacex((unsigned char)*str))
            *ret += 0x100;              /* スペース類につける重み */
        return *str ? str+1 : NULL;
    }
    return NULL;
}

/*
 *  文字列の比較処理
 */
int ecomp(const char *s1, const char *s2)
{
    int i;
    const char *p1, *p2;
    unsigned code1, code2;

    /* まったく一致しているかどうかを検査する */
    if (strcmp(s1, s2) == 0) return 0;

    /* 基本的な比較から詳細な比較へ、段階的に比較処理を実行する */
    for (i = ECHAR_PHASE_1; i <= ECHAR_PHASE_4; i++) {
        p1 = s1;
        p2 = s2;
        do {
            p1 = echar(i, p1, &code1);
            p2 = echar(i, p2, &code2);
            if (code1 == 0 && code2 == 0) break;
            if (code1 == 0) return -1;
            if (code2 == 0) return 1;
            if (code1 != code2) return code1 - code2;
        } while (p1 || p2);
    }
    return 0;
}
```

ままでは'b'の文字コードが'+'の文字コードよりも大きいためfoo+bar<foobarとなってしまいます．このように，重みをつけることで新たに区別の対象となる文字を正しく評価できます．段階の移行で有効になる文字は1文字にかぎりませんが，ここではそれぞれの文字コードがそのまま評価されます．たとえば，第2段階ではfoo#bar<foo+barとなります．

日本語の辞書順ソート

日本語の文字列の比較には，かなり複雑な規則を適用しなければなりません．かな文字だけでも，濁音･半濁音･促音・拗音・長音などを考慮しなければなりません．さらに，カタカナ・ひらがな，半角文字・全角文字などの違いがあります．英字が含まれる可能性もあるわけですから，先ほどの英字ソートの規則を無視することもできません．

日本語については，三省堂の『新明解国語辞典』の規則と前述の英字ソートの規則を組み合わせて，次のような規則で比較を行います．

日本語の比較規則

①最初は，英数字・かな・漢字にのみ注目して比較する．英字の大・小文字の区別はしない．かな文字ついては，カタカナ・ひらがな，濁音・半濁音・促音・拗音などの区別はしない．これらは，英数字<かな<漢字の順に並べられる．また，すべての文字について全角・半角の区別をしない．長音記号(「ー」)については，1つ前の文字がカナ文字であれば対応する発音(ア・イ・ウ・エ・オのいずれか．ただし，1つ前がンの場合にはン)に置き換えられる．

②①では文字列が一致してしまう場合，記号を含めて比較する．ある位置で片方の文字列にのみ記号があらわれた場合は，記号のないものよりも大きい(つまり後ろ)ものとする．ここで，スペース，ドット(省略記号として使われる)，ハイフン(文字の区切りとして使われる)についてはスペース類文字として比較の対象としない．

③②では文字列が一致してしまう場合，かな文字の清音(「ハ」など)・濁音（「バ」など)･半濁音（「パ」など）を区別して比較する．比較順序は，清音→濁音→半濁音となる．

④③では文字列が一致してしまう場合，カナ文字の促音（「っ」)･拗音(「ゃ」など）を区別して比較する．比較順序は，促音・拗音→直音(「つ」「や」など）となる．

⑤④では文字列が一致してしまう場

リスト19 (ectest.c)

```
/*
 *  ectest.c: ecomp 関数の使用例
 */
#include <stdio.h>
#include <stdlib.h>

/* 英数文字列を比較する関数のプロトタイプ */
int ecomp(const char *s1, const char *s2);

/* 文字列リストの例 */
char *table[] = {
    "screwed",  "sand",     "South Africa",
    "screw",    "S.O.S.",   "San Francisco",
    "screw-up", "software", "Saturday",
};

#define TBLMAX (sizeof table / sizeof table[0])

/* qsort のための比較関数 */
int compare(const char **s1, const char **s2)
{
    return ecomp(*s1, *s2);
}

/* 文字列リストの内容を表示する */
void dsptable(void)
{
    int i;

    printf("-- string table --¥n");
    for (i = 0; i < TBLMAX; i++)
        printf("table[%d] = %s¥n", i, table[i]);
}

void main()
{
    dsptable();
    /* ソートの実行 */
    qsort(table, TBLMAX, sizeof (char *),
        (int (*)(const void *, const void *))compare);
    dsptable();
}

```

図1 英字のソート例

```
-- string table --
table[0] = screwed
table[1] = sand
table[2] = South Africa
table[3] = screw
table[4] = S.O.S.
table[5] = San Francisco
table[6] = screw-up
table[7] = software
table[8] = Saturday
-- string table --
table[0] = sand
table[1] = San Francisco
table[2] = Saturday
table[3] = screw
table[4] = screwed
table[5] = screw-up
table[6] = software
table[7] = S.O.S.
table[8] = South Africa
```


ワンラインNEWS
HAL研究所は，シャープの電子手帳「PA9500(DB-Z)」とパソコンを接続するためのインターフェース「Hyper HAL-CATCH」を発売．日本電気のPC-9801に対応．

合，カタカナ・ひらがなを区別して比較する．比較順序は，カタカナ→ひらがなとなる．

⑥⑤では文字列が一致してしまう場合，英字の大小文字を区別して比較する．比較順序は，英小文字→英大文字となる．

⑦⑥では文字列が一致してしまう場合，ある位置で片方の文字列にのみスペース類文字があらわれた場合は，スペース類文字のないものよりも大きい（つまり後ろ）ものとする．

⑧⑦では文字列が一致してしまう場合，全角･半角を区別する．比較順序は，半角文字→全角文字となる．

⑨以上で2つの文字列に違いがなければ，両者を一致するものとみなす．

この規則でプログラミングした例がリスト20（jcomp. c）です．使用例をリスト21（jctest. c）に示します．

リスト21の実行結果は図2のようになります．英字ソートの例と同じように，ソート前とソート後の配列tableの内容が表示されています．

リスト20 （jcomp.c）

```c
/*
 *  jcomp.c: 日本語文字列の比較
 *      written by Motohisa Ohno.
 */
#include <stddef.h>
#include <ctype.h>
#include <string.h>
#include <jctype.h>
#include <jstring.h>

enum jchar_phase_t {
    JCHAR_PHASE_1,  /* 英数字（大小文字を無視）、カナ（基本音のみ）、漢字 */
    JCHAR_PHASE_2,  /* 英数字、カナ、漢字、記号（スペース類以外） */
    JCHAR_PHASE_3,  /* 英数字、カナ（濁音・半濁音を区別）、漢字、記号 */
    JCHAR_PHASE_4,  /* 英数字、カナ（促音・拗音を区別）、漢字、記号 */
    JCHAR_PHASE_5,  /* 英数字、カナ（カタカナ、ひらがなを区別）、漢字、記号 */
    JCHAR_PHASE_6,  /* 英数字（大小文字を区別）、カナ、漢字、記号 */
    JCHAR_PHASE_7,  /* 英数字、カナ、漢字、記号（スペース類を含む） */
    JCHAR_PHASE_8,  /* 全角・半角を区別 */
};

enum jchar_t {
    JCHAR_UPPER = 0x0001,
    JCHAR_LOWER = 0x0002,
    JCHAR_DIGIT = 0x0004,
    JCHAR_ALNUM = JCHAR_UPPER | JCHAR_LOWER | JCHAR_DIGIT,

    JCHAR_SPACEX = 0x0008,
    JCHAR_KIGOU = 0x0010,
    JCHAR_CNTRL = 0x0020,

    JCHAR_HIRA = 0x0040,
    JCHAR_KATA = 0x0080,
    JCHAR_KANA = JCHAR_HIRA | JCHAR_KATA,

    JCHAR_DAKU = 0x0100,        /* 濁音 */
    JCHAR_HANDAKU = 0x0200,     /* 半濁音 */
    JCHAR_SOKU = 0x0400,        /* 促音・拗音 */

    JCHAR_KANJI = 0x0800,       /* 漢字 */

    JCHAR_ZEN = 0x1000,         /* 全角 */
    JCHAR_HAN = 0x2000,         /* 半角 */
};

struct jchar_s {
    unsigned code;      /* 文字コード */
    unsigned type;      /* 英大・小文字の区別、カナの濁音・促音など */
    int weight;         /* 重み */
};

/*
 *  文字コードに対応する母音
 */
static unsigned char kana_boin_tbl[] =
    "アアイイウウエエオオアアイイウウ"
    "エエオオアアイイウウエエオオアア"
    "イイウウウエエオオアイウエオアア"
    "アイイイウウウエエエオオオアイ  "
    "ウエオアアウウオオアイウエオアア"
    "イエオンウアエ";

#define getboin(code) ((kana_boin_tbl[((code) - 0x8340)*2] << 8) ¥
    | kana_boin_tbl[((code) - 0x8340)*2+1])

/*
 *  文字コードに対応する清音
 */
static unsigned char kana_seion_tbl[] =
    "アアイイウウエエオオカカキキクク"
    "ケケココササシシススセセソソタタ"
    "チチツツツテテトトナニヌネノハハ"
    "ハヒヒヒフフフヘヘヘホホホマミム"
    "ムメモヤヤユユヨヨラリルレロワワ"
    "キエヲンウカケ";

#define getseion(code) ((kana_seion_tbl[((code) - 0x8340)*2] << 8) ¥
    | kana_seion_tbl[((code) - 0x8340)*2+1])

/*
 *  文字コードに対応するフラグ
 */
#define J0  0
#define JD  JCHAR_DAKU
#define JH  JCHAR_HANDAKU
#define JS  JCHAR_SOKU

static unsigned kana_flag[] = {
    JS, J0, JS, J0, JS, J0, JS, J0, JS, J0, J0, JD, J0, JD, J0, JD,
    J0, JD, J0, JD, J0, JD, J0, JD, J0, JD, J0, JD, J0, JD, J0, JD,
    J0, JD, JS, J0, JD, J0, JD, J0, JD, J0, J0, J0, J0, J0, J0, JD,
    JH, J0, JD, JH, J0, JD, JH, J0, JD, JH, J0, JD, JH, J0, J0, J0,
    J0, J0, J0, JS, J0, JS, J0, JS, J0, J0, J0, J0, J0, J0, JS, J0,
    J0, J0, J0, J0, JD, JS, JS,
};

#undef JS
#undef JH
#undef JD
#undef J0

/*
 *  文字列が指す文字種を詳しく解析する
 */
#define isspacex(x) ¥
    (isspace(x) || (x) == '－' || (x) == '＿' || (x) == '．' || (x) == 0xa0)

/*
 *  半角カナを全角カナに変換するためのテーブル
 */
static unsigned char han2zen_kana_tbl[] =
    " 。「」、・ヲァィゥェォャュョッ"
    "ーアイウエオカキクケコサシスセソ"
    "タチツテトナニヌネノハヒフヘホマ"
    "ミムメモヤユヨラリルレロワン゛゜";

static const char *analyze(const char *str,
                        struct jchar_s *sp, struct jchar_s *lastp)
{
    unsigned code;

    code = (unsigned char)*str;
    ++str;
    if (iskanji(code)) {        /* 2バイト文字 */
        code = (code << 8) | (unsigned char)*str;
        ++str;
        if (jisupper(code)) {
            sp->type = JCHAR_UPPER;
        } else if (jislower(code)) {
            sp->type = JCHAR_LOWER;
            code = jtoupper(code);
        } else if (jisdigit(code)) {
            sp->type = JCHAR_DIGIT;
        } else if (jiskata(code)) {
            sp->type = JCHAR_KATA | kana_flag[code - 0x8340];
            code = getseion(code);
        } else if (jishira(code)) {
            code = jtokata(code);
            sp->type = JCHAR_HIRA | kana_flag[code - 0x8340];
            code = getseion(code);
        } else if (0x889F <= code) {    /* 漢字（外字を含む） */
            sp->type = JCHAR_KANJI;
        } else {
            switch (code) {
            case 0x815B:        /* 長音 */
                if (lastp->type & JCHAR_KANA) {
                    code = getboin(lastp->code);
                    sp->type = lastp->type;
                } else {
                    sp->type = JCHAR_KIGOU;
                }
                break;
            case 0x8152:        /* ヽ */
            case 0x8154:        /* ゝ */
                if (lastp->type & JCHAR_KANA) {
                    code = lastp->code;
                    sp->type = lastp->type;
                } else {
                    sp->type = JCHAR_KIGOU;
                }
                break;
            case 0x8153:        /* ヾ */
            case 0x8155:        /* ゞ */
                if (lastp->type & JCHAR_KANA) {
                    code = lastp->code;
                    sp->type = lastp->type | JCHAR_DAKU;
                } else {
                    sp->type = JCHAR_KIGOU;
                }
                break;
            case 0x8158:        /* 々 */
                if (lastp->type & JCHAR_KANJI) {
                    code = lastp->code;
                    sp->type = JCHAR_KANJI;
                } else {
                    sp->type = JCHAR_KIGOU;
                }
                break;
            case 0x8140:        /* スペース */
```

ワンラインNEWS
UNIXシステムラボラトリーズパシフィックは，UNIX SystemVリリース4（SVR4）の日本語解説書を発売した．ユーザー向けに書店を通して販売する．

```
            case 0x817c:        /* ー */
            case 0x8151:        /* _ */
            case 0x8144:        /* . */
                sp->type = JCHAR_SPACEX;
                break;
            default:
                sp->type = JCHAR_KIGOU;
            }
        }
        sp->code = code;
        if ((sp->type & JCHAR_KANA) && ((unsigned char)*str == 0x81)) {
            /* 濁音(0x814A)、半濁音(0x814B)の判定 */
            if ((unsigned char)*(str+1) == 0x4A) {
                sp->type |= JCHAR_DAKU;
                str += 2;
            } else if ((unsigned char)*(str+1) == 0x4B) {
                sp->type |= JCHAR_HANDAKU;
                str += 2;
            }
        }
        sp->type |= JCHAR_ZEN;
        return str;
    }
    if (!isprkana(code)) {  /* 制御文字 */
        sp->code = code;
        sp->type = isspace(code) ? JCHAR_SPACEX : JCHAR_CNTRL;
        sp->type |= JCHAR_HAN;
        return str;
    }
    if (isprint(code)) {    /* 0x20 <= code <= 0x7E */
        if (isupper(code)) {
            sp->type = JCHAR_UPPER;
        } else if (islower(code)) {
            sp->type = JCHAR_LOWER;
            code = _toupper(code);
        } else if (isdigit(code)) {
            sp->type = JCHAR_DIGIT;
        } else if (isspacex(code)) {
            sp->type = JCHAR_SPACEX;
        } else {
            sp->type = JCHAR_KIGOU;
        }
        sp->code = hantozen(code);
    } else {                    /* 0xA1 <= code <= 0xDF */
        if (code == 0xB0) {             /* 長音 */
            if (lastp->type & JCHAR_KANA) {
                sp->code = getboin(lastp->code);
                sp->type = lastp->type;
                return str;
            }
            sp->type = JCHAR_KIGOU;
        } else if (code <= 0xA5) {      /* カナ記号 */
            sp->type = JCHAR_KIGOU;
        } else if (code <= 0xDD) {      /* カナ文字 */
            sp->type = JCHAR_KATA;
            if (code <= 0xAF)           /* 促音・拗音 */
                sp->type |= JCHAR_SOKU;
        } else {                        /* 単独の濁音、半濁音 */
            sp->type = JCHAR_KIGOU;
        }
        code = (code - 0xA0) * 2;
        sp->code = (han2zen_kana_tbl[code] << 8) | han2zen_kana_tbl[code+1];
        if ((unsigned char)*str == 0xDE) {  /* 濁音 */
            sp->type |= JCHAR_DAKU;
            ++str;
        } else if ((unsigned char)*str == 0xDF) {   /* 半濁音 */
            sp->type |= JCHAR_HANDAKU;
            ++str;
        }
    }
    sp->type |= JCHAR_HAN;
    return str;
}

#pragma argsused

const char *jchar(int phase, const char *str,
                  struct jchar_s *sp, struct jchar_s *lastp)
{
    if (str == NULL) {
        sp->code = 0;
        sp->weight = 0;
        return NULL;
    }

    switch (phase) {
    case JCHAR_PHASE_1:
        /* 英数字（大小文字を無視）、カナ（基本音のみ）を返す */
        while (*str) {
            str = analyze(str, sp, lastp);
            if (sp->type & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI)) {
                sp->weight = 0;
                return *str ? str : NULL;
            }
        }
        break;

    case JCHAR_PHASE_2:
        /* 英数字、カナ、記号（スペース類以外） */
        /* 記号なし→記号ありの順 */
        while (*str) {
            str = analyze(str, sp, lastp);
            if (sp->type
                    & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI | JCHAR_KIGOU)) {
                sp->weight = (sp->type & JCHAR_KIGOU) ? 1 : 0;
                return *str ? str : NULL;
            }
        }
        break;

    case JCHAR_PHASE_3:
        /* 英数字、カナ（濁音・半濁音を区別）、記号 */
        /* 清音→濁音→半濁音の順 */
        while (*str) {
            str = analyze(str, sp, lastp);
            if (sp->type
                    & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI | JCHAR_KIGOU)) {
                sp->weight = (sp->type & JCHAR_HANDAKU) ? 2
                    : (sp->type & JCHAR_DAKU) ? 1 : 0;
                return *str ? str : NULL;
            }
        }
        break;

    case JCHAR_PHASE_4:
        /* 英数字、カナ（促音・拗音を区別）、記号 */
        /* 促音・拗音→直音の順 */
        while (*str) {
            str = analyze(str, sp, lastp);
            if (sp->type
                    & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI | JCHAR_KIGOU)) {
                sp->weight = (sp->type & JCHAR_SOKU) ? 0 : 1;
                return *str ? str : NULL;
            }
        }
        break;

    case JCHAR_PHASE_5:
        /* 英数字、カナ（カタカナ、ひらがなを区別）、記号 */
        /* カタカナ→ひらがなの順 */
        while (*str) {
            str = analyze(str, sp, lastp);
            if (sp->type
                    & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI | JCHAR_KIGOU)) {
                sp->weight = (sp->type & JCHAR_HIRA) ? 1 : 0;
                return *str ? str : NULL;
            }
        }
        break;

    case JCHAR_PHASE_6:
        /* 英数字（大小文字を区別）、カナ、記号 */
        /* 小文字→大文字の順 */
        while (*str) {
            str = analyze(str, sp, lastp);
            if (sp->type
                    & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI | JCHAR_KIGOU)) {
                sp->weight = (sp->type & JCHAR_UPPER) ? 1 : 0;
                return *str ? str : NULL;
            }
        }
        break;

    case JCHAR_PHASE_7:
        /* 英数字、カナ、記号（スペース類を含む） */
        str = analyze(str, sp, lastp);
        sp->weight =
            (sp->type & (JCHAR_ALNUM | JCHAR_KANA | JCHAR_KANJI | JCHAR_KIGOU))
                ? 0 : 1;
        return *str ? str : NULL;

    case JCHAR_PHASE_8:
        str = analyze(str, sp, lastp);
        sp->weight = (sp->type & JCHAR_ZEN) ? 1 : 0;
        return *str ? str : NULL;
    }

    sp->code = 0;
    sp->weight = 0;
    return NULL;
}

/*
 *  文字列の比較処理
 */
int jcomp(const char *s1, const char *s2)
{
    int phase;
    const char *p1, *p2;
    struct jchar_s js1, jlast1;
    struct jchar_s js2, jlast2;

    /* まったく一致しているかどうかを検査する */
    if (strcmp(s1, s2) == 0) return 0;

    jlast1.code = 0;
    jlast1.type = 0;
    jlast2 = jlast1;

    for (phase = JCHAR_PHASE_1; phase <= JCHAR_PHASE_8; phase++) {
        p1 = s1;
        p2 = s2;
        for (;;) {
            p1 = jchar(phase, p1, &js1, &jlast1);
            p2 = jchar(phase, p2, &js2, &jlast2);

            if (js1.code == 0 && js2.code == 0) break;
            if (js1.code == 0) return -1;
            if (js2.code == 0) return 1;
            if (js1.weight != js2.weight)
                return js1.weight - js2.weight;
            if (js1.code > js2.code)
                return 1;
            if (js1.code < js2.code)
                return -1;
            jlast1 = js1;
            jlast2 = js2;
        }
    }
    return 0;
}
```


ワンラインNEWS
日本電気ホームエレクトロニクスは，現在の冷陰極型蛍光ランプより６割明るく，寿命は10倍以上のOA機器用ランプを開発．ワープロのバックライト向けに使える．

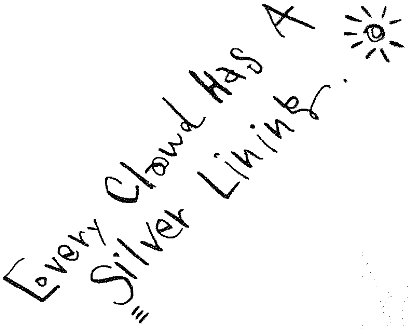
プログラムの解説

ecomp関数と同様にしてjcomp関数を作成します．ここでは，同一の文字に対しさまざまな文字種が存在するため，それらを解析するためのanalyze関数を用意しています．たとえば，「つ」という文字に対しては半角のﾂ・ｯ・ﾂﾞ，全角のツ・ッ・ヅ・つ・っ・づが存在します．

analyze関数は，文字列が示す文字コード，文字種を返します．analyze関数には，長音（ー）などを処理するため，1つ前の文字に関する情報を渡さなければなりません．文字コードは，等価な全角文字コード（かなはすべてカタカナの清音）として，構造体jchar_sのメンバcodeに渡されます．文字種に関する情報は，メンバtypeに渡されます．前述の「つ」の例であれば，メンバcodeはすべて全角の「ツ」となり，それぞれの文字種に応じてメンバtypeが設定されます．

jchar関数は，analyze関数で解析された内容をもとに，比較段階にしたがって該当する文字コードを返します．文字コードが1ワードを占めるため，echar関数で文字コードの上位バイトに与えていた重みをメンバweightに与えます．jcomp関数とjchar関数の関係は，ecomp関数とechar関数の関係とほぼ同じです．

日本語をソートするために，1回の文字列比較で最高8段階もの繰り返しが行われるため，かなり効率が悪くなるように見えます．実際，似たような文字列で構成される数100個の文字列リストで試したところ，ソートするのに数秒かかりました．実際の文字列は，最初の1，2段階で区別できることが多いでしょうが，項目数がかなり多くなる場合は，先頭の文字をanalyze関数で解析させ区別しておく必要があるかもしれません．ソート対象がメモリでない場合には，さらに工夫する必要があるでしょう．



リスト21 （jctest.c）

```c
/*
 *  jctest.c: jcomp 関数の使用例
 */
#include <stdio.h>
#include <stdlib.h>

/* 日本語文字列を比較する関数のプロトタイプ */
int jcomp(const char *s1, const char *s2);

/* 文字列リストの例 */
char *table[] = {
    "きやく",      "規約",
    "きゃく",      "客",
    "ぎゃく",      "逆",
    "ギャグ",      "ｷﾞｬｸﾞ",
    "きや",        "木屋",
    "ギヤ",        "ｷﾞﾔ",
    "ぎゃくさん",  "逆算",
    "きゃくしつ",  "客室",
};

#define TBLMAX (sizeof table / sizeof table[0])

/* qsort のための比較関数 */
int compare(const char **s1, const char **s2)
{
    return jcomp(*s1, *s2);
}

/* 文字列リストの内容を表示する */
void dsptable(void)
{
    int i;

    printf("-- string table --¥n");
    for (i = 0; i < TBLMAX; i++)
        printf("table[%d] = %s¥n", i, table[i]);
}

void main()
{
    dsptable();
    /* ソートの実行 */
    qsort(table, TBLMAX, sizeof (char *),
        (int (*)(const void *, const void *))compare);
    dsptable();
}
```

図2 日本語のソート例

```
-- string table --
table[0] = きやく
table[1] = 規約
table[2] = きゃく
table[3] = 客
table[4] = ぎゃく
table[5] = 逆
table[6] = ギャグ
table[7] = ｷﾞｬｸﾞ
table[8] = きや
table[9] = 木屋
table[10] = ギヤ
table[11] = ｷﾞﾔ
table[12] = ぎゃくさん
table[13] = 逆算
table[14] = きゃくしつ
table[15] = 客室
-- string table --
table[0] = きや
table[1] = ｷﾞﾔ
table[2] = ギヤ
table[3] = きゃく
table[4] = きやく
table[5] = ぎゃく
table[6] = ｷﾞｬｸﾞ
table[7] = ギャグ
table[8] = ぎゃくさん
table[9] = きゃくしつ
table[10] = 規約
table[11] = 客
table[12] = 客室
table[13] = 逆
table[14] = 逆算
table[15] = 木屋
```

98活用アセンブラ技法

番外編
第2回

書式およびデバックのコツ

星野 操

アセンブラでプログラムを作成する場合，いくつか留意すべき点があります．ここでは，コーディングの際のインデントのつけ方，ラベルの命名法など，アセンブラの書式について述べるとともに，デバッガを使用したアセンブラプログラムのデバッグ法について解説します．

ライブラリ

できるだけ既存の情報を利用するのが利口

今では，大きなプログラムを頭から最後までアセンブラで書くということは，あまりありません．そのようなプログラムの場合，バリエーションにある程度の制限が出てきます．たとえば，デバイスドライバを書く場合，常駐プログラムを書く場合，sjis⟷jisのコード変換を書くなど速度が要求される場合，などなど．

デバイスドライバや常駐プログラムは，実際に目的を達する手続き以外に，各種のおまじないが必要になります．デバイスドライバならデバイスドライバヘッダ，パケットの構造体定義，各種エントリポイントの処理などです．常駐プログラムなら，常駐サイズ計算部分や常駐実行部分など．これらは，目的によらずほぼ同一のものが使用されます．このようなプログラムは「一度作っておけば」ちょっと変わったものを作るのは比較的容易で，部分的な流用がやりやすい分野といえます．したがって，このようなプログラムは積極的にライブラリとして作成しておくべきです．自分で書けばもちろん最高ですが，手もちのバリエーションに入っていなければ雑誌に載っていたものでもかまいません．BBSなどのネットワークで流れているソースつきのプログラムであればなおよいでしょう．

人の書いた（それも動くことが期待できる）プログラムは，似たようなものを作るときにおおいにヒントを与えてくれます．どうしてもとれないバグに悩んでいるときに他人のソースを読んで，自分が知らなかったり見落としていたおまじないを発見し，目から鱗が落ちるような思いをすることも少なくありません．問題点がわかれば，修正するのはいとも簡単です．

もちろん，そのような他人のプログラムを完全に引き写してしまうと，いろいろ権利の問題が生じますから，「ヒント（アイデアややり方）」だけを参考にするに留めておきましょう．とくに権利関係が重要になってくるようなプログラムでは，他人のプログラムを参照することすら好ましくないかもしれません（そのようなポリシーをもち，なるべく他人のプログラムを目にしないように努力している人もいます）．しかし，自分のためのプログラムを作るなら，けっしてひけ目を感じることはないのです．そしてその技が完全に自分のものになってしまえば（そこまで抽象化されてしまえば），だれにも文句をいわれる筋合いではないと思います．

プログラムと同様に，プリントされた各種資料もきわめて重要です．単行本，ムック，雑誌記事などは「必要なときに」取り出せるように，その内容の存在と物理的な存在位置を確認しておく必要があるでしょう．

プログラムなんて，何から何まで自分1人でできるものではありません．それなら，できるだけ既存の情報を利

ワンラインNEWS
アールブルーは企業経営の健全性や倒産の危険度を評価できる経営分析ソフト「MR・企画部帳」を開発した．中小企業のオーナーや金融業者に売り込む．

用するのが利口ではないでしょうか．それでこそ，先人の苦労が報われるというものです．

書式

アセンブラでは必要のないスペースを入れないのが基本

次に，アセンブラのプログラムの書き方についてちょっと考えてみましょう．通常，アセンブラでは8カラムのタブスペースが使用されます．Cなどでは4カラムタブが通常ですし，Pascalなどでは2カラムも使用されます．

Pascalではbegin-endのインデントの書き方にもよりますが，一般にインデントが多重につけられ非常に深くなるので2カラムタブになるようです．Cではまれに8カラムで書く人もあるようですが，これだと少しインデントが深くなると右端にはみだしてしまいます．このため，4カラムタブが一般的に使用されるようです．

アセンブラでは，構造化マクロを使用するような場合は別ですが，基本的にはインデントがつけられることはありません．このため，タブ幅が大きくてもあまり問題はないのです．4カラムにしないのは，アセンブラの命令の一部が4文字長以上であり，4カラムにするとオペランドの位置が揃わなくなるからです．

オペコードとオペランドをあまり離してしまうと可読性が悪くなります．昔の6800CPUのように，すべての命令が3文字として規定されていれば4カラムタブで使用することもできたでしょうが，やはり現在のアセンブラでは8カラムタブで使用するのが無難であるといえます．

日本のほとんどのエディタは「ハードウェアタブ」を使用しています．たとえば，筆者がこの原稿を書いているVzエディタでも，日本でおそらくもっとも使われているエディタであるMifesでも，デフォルトの設定はハードウェアタブです．これに対して，海外のエディタではほとんどがタブをスペースに展開しています．

ハードウェアタブだと，文字列をエディットしたあとのタブ位置もつねに正規化されるというメリットがあります．しかし，プリンタによってはプリントアウトするときに障害が生じますし，プログラムの作成者が「何カラムタブ」でプログラムを書いたか，というのが問題になってきてしまいます．

たとえば，あるアセンブラのプログラムに，明らかに作者は4カラムのハードタブを使用して，かつ見かけ上8カラムタブになるように設定してあるものがあります．つまり，ADDのあとには2タブを入れ，INSBのあとには1タブ入れてあるわけです．

このようなソースを通常の8カラムタブの設定で見ると，異常に右にシフトして，またオペランドも揃わなくなり，たいへん見にくくなります．8カラムタブで印刷したものなど，ほとんど読むことはできません．

このように標準的な記法と異なる書き方をしていると，変換動作をかまさ

ワンラインNEWS
マイクロヘリオスは米サン・マイクロシステムズ社のワークステーション「SPARCstation」の互換機を発売．サンの製品より価格を安く設定することで市場への浸透を図る．

ないとソースを読むことすらままならなくなります（印刷してあればそのようなこともできません）．ハードウェアタブを使用する場合に，8カラムタブという基本を外すことは賢明とはいえないでしょう．

一方，アメリカのエディタでよく使用されるソフトウェアタブでは，以上のような問題はいっさい生じません．ただし，タブがスペースに展開されていると「タブをもどす」動作をするときに多少めんどうな場合があります．たとえば，8カラムタブだと[BS]を8回押さないとタブを消すことができません．しかし，このようなエディタでは[SHIFT]+[TAB]でこのバックタブ動作を1回でできるのが普通ですから，これもそれほどの難点とはいえません．もし難点があるとすれば，修正を入れたときに後ろのタブ位置がずれてしまうということでしょうか．しかし，これにしても十分に対処可能です．ソフトウェアタブの使用は，もっと一般的になってもよいと思います．

オペコードとオペランドの間だけではなく，オペコードの前にもタブを入れます．これも普通は1タブだけですが，稀に2タブを入れる人もいます．86のアセンブラでは，ラベルのあとにオペコードを続けてもかまいません．アドレスラベルではオペコードが続く必要はないのですが，86アセンブラ特有の変数ラベルの場合には，あとにdbなどの疑似命令が続きます．

このラベルの名前が長いと，1タブでは後ろのオペコードの位置がずれてしまうことになります．アドレスラベルの場合で6文字，変数ラベルの場合で7文字を超えるとこうなります．それを嫌って2タブを入れているのでしょう．しかし，これはなるべくオペコードをラベルのあとに続けない，ということで回避できる場合が多いのです．

ほとんどのラベルは分岐先に使用されるだけですから，これのあとにオペコードを続けない，つまりアドレスラベルはアドレスラベルとして独立させればいいわけです．また，このようにラベルで流れを中断することによって，そこから分岐合流するということを明示的に示すことができる効用もあります．変数ラベルではこううまくはいきませんが，プログラム中で変数の使われる割合は，ずっと少ないでしょうからなんとか我慢できるでしょう．

結局，

・8カラムタブを使用．オペコードの前に1タブ，オペコードとオペランドの間に1タブ．
・アドレスラベルのあとには命令コードを続けない．

という一般的な記法に合わせるのがいちばんです．

アセンブラでも構造化マクロなどを使用すれば，実際問題としてアドレスラベルはほとんど不要になります．オペランドとオペコードの間には8カラムのタブが必要ですが，このまま使用すると多少インデントが深くなり過ぎ（Cと同様に）ます．かといってタブ幅を小さくするわけにもいきませんから，マクロやサブルーチンを積極的に使用して，インデントが深くなり過ぎないようにするべきです．

もう1つ注意するのは，スペースの使い方です．Cなどでは，二項演算子・代入演算子などの両端には1スペースあけるというのが定石です．これは，古い代入記法を避ける，コメント文字の開始記号が現われるのを避ける，見やすさなどの理由によるものです．Cほどスペースをあける必然性のないほかの言語であっても，やはりスペースをあけたほうが見やすくなるのでそのようにするべきでしょう．

しかし，アセンブラはCとは異なり，必要のないスペースを使用してはいけない，というのが基本です．なぜなら，アセンブラではスペースは往々にして区切り記号と判断されてしまうからです．これは筆者もなかなか守れずにいます．

通常の組み込みオペコードや疑似命令のオペランドとして使用する場合は，スペースをあけてもなんら問題はないのですが，マクロの引数ではスペースが「,」と同様に引数の区切り記号になってしまいます．たとえば，マクロの引数に，

```
BUFSIZE  + 1, ax
```

というようなものを渡すと，第1引数が〈BUFSIZE + 1〉，第2引数が〈ax〉になるのではなく，第1引数がBUFSIZE，第2引数が＋というように4つに分解されてしまいます．これを避けるには，全体を<>で括らなければなりません．しかし，これを忘れてしまうことがよくあるのです．アセンブラでは，スペースを使用しなくてよい場面では，スペースを入れないという書式を守ったほうが安全です．とくに，マクロを多用する場合には注意が必要です．もちろん，SEGなど記号でない文字列で構成される演算子を使用する場合には，スペースをあけないわけにはいきません．この場合は，忘れずに<>で括ってやらなければいけません．またマクロの引数は必ず<>で括ってやるようにして，引数の分解はirpでやるようにすれば，このスペースの

ワンラインNEWS
ライオスは日本IBMの「PS/55note」に接続し，PC/AT用LANボードなどを収納できる拡張スロット「Note Box」の販売を開始した．SCSIに対応した周辺機器も接続できる．

問題を回避することも可能です．

命名法

自分なりの決まりを作って プログラム内で統一をとる

オペコード・レジスタ名は，でき合いのものを使用するしかありません．もちろん，マクロでこれらを置き換えることはできます．しかし，それは明らかに可読性を悪くするでしょう．セグメント名についても，とくに高級言語とのインターフェースの関係から，標準的なものが定められています．本連載で使用したような簡易セグメント指定を使用すれば，無条件でセグメント名は決まってしまいます．ですから，これらでまにあう場合には，これら標準的な名前を使用するべきでしょう．

これに対して，ジャンプ先を示すアドレスシンボル，変数ラベル，マクロ名やローカル変数名などには自由に名前がつけられます．これらの命名法についても，自分なりに決まりを作っておくとよいでしょう．

まずジャンプ先のラベルですが，これは関数名(proc名)でなければきわめて便宜的なものですから，シンボル自体を使わないようにするのがいちばんです．構造化マクロを使用する手がありますし，アセンブラによって実装されているローカルシンボルを使用するという手もあります．後者は，MASMのそれよりはTASMのそれのほうが使いでがあったと思います．

場合によっては，直接オフセットを書きたくなることもあります．たとえば条件のロングジャンプをする場合には，補条件と条件ジャンプを組み合わせるのが普通です．このときに，条件ジャンプの飛び先を＄＋nの形で表記すれば，ローカルシンボルを使用しなくてすみます．ただ，飛び越す内容が固定長命令（たとえばSTCなど）ならよいのですが，条件ロングジャンプの場合は，下手をすると飛び越す命令の長さが変わってしまいます．前方参照の場合は，ジャンプの最適化がされていなければ3バイト長と考えてよいのですが，後方参照の場合はレンジによって長さが変わります．ですから，たとえば短いレンジを仮定してjb $+4などと書いておくと，命令コードの追加によってあとの条件ジャンプが3バイトになったとたんに暴走します．このような場合には，必ず無条件ジャンプのオペランドにnear ptrの修飾子をつけ，3バイト長として使う必要があります．

次に変数名ですが，これは高級言語で変数名をつける場合と同様です．基本的にはローカル変数などは短い名前で，大域変数などには長めの理解しやすい名前をつけることが推奨されます．もちろん，各種レジスタとコンフリクトしそうな名前をつけてはいけません．

長い名前は，フルスペルの英文句で記述するのが基本です．単語の区切りにはアンダースコアを使う方法と単語の先頭を大文字にする方法とがあります．かなり前のアセンブラでは，大文字小文字の区別がありませんでした．現在でも，これはオプションになっています．安全性からいうなら，アンダースコアで区切るほうがよいといえるでしょう．ただ，高級言語もだんだん大文字小文字で区切るようになってきていますし（ハンガリアンまではいかなくても），整合性を考えるとアンダースコアでないほうがよいかもしれません．

ワンラインNEWS
リブネットはアップル社の「Macintosh」を使って相手の動画像を見ながら会話のできる通信システムを開発．従来のテレビ会議システムの1/10程度のコストで実現できる．

変数名も大昔には長さの制限がありました．内部的にはどれだけ長くてもよかったのですが，.objに書き出すときに有効になるシンボル長に制限があり，このためグローバルに有効なシンボルには実質的に長さの制限があったのです．このため，変数名・関数（アドレスラベル）名には，省略された単語の記述（たとえばbuffer-> buf）が使用され，同じような名前が複数ある場合には，変化のある部分を先にもってくる（たとえば，read_buffer1，read_buffer2ではなく，buf1_read，buf2_readというふうに）という「決まり」がありましたが，現在ではそういう意味での制限はなくなっています．ですから，なるべく自然な命名法が好ましいといえます．

ある程度わかりやすい名前であれば，プログラムを「読む」うえではどのような表記であろうとあまり問題はありません．しかし，書く段（修正する段も含む）になると話が違ってきます．たとえば，bufferという意味でbufとかbuffという省略記法が広く使われています．strbuf(string buffer)とかiobuff (I/O buffer) などのように，読む分には，十分に意図が理解できますが，書くときに果たして目的とする変数はどの記法で書いてあったかというのがわからなくなります．これらの省略法が混じっていたりすると最悪です．結局，プログラムの中をスキャンし，目的とする変数の実際の定義を確認したりするはめになります．

アンダースコアによる区切りの有無についても同様です．また，動詞句を意味する関数名などの場合，動詞＋名詞で記述したか名詞＋動詞で記述したか（read_bufferかbuffer_readか）が混乱するとかなりパニックになります．ですからプログラムを書く場合には，これらの自由度をある程度制限したほうが結果として楽になるのです．実際には先に書いたように，省略記法は使わない（これによって，単語は一意に記述される），区切りの仕方は統一する，かつ必ず実行するようにする，英表記に従い，動詞＋名詞などの構成を基本とする，などを守るようにするわけです．

ただし，これらが原則論であることは否定できません．筆者も高級言語ほどにはこれらの原則を守っているとはいえません（高級言語ですら完全とはいえず……）．アセンブラはとくにプログラムの記述量が多くなるため，変数名などに過剰に長い表記をし，タイプ数をより多くすることは好まれません．このため，短い省略記法が好んで使われる傾向があります．ニーモニック自体が省略記法ですし，segとかptrとかprocとか疑似命令やそのほかのオペレータにも省略記法がバンバン出てきます．

さらに，アンダースコアなどによる区切りを原則としても，Cに存在するatoiとかstrcmpなどに相当するプログラムを書いた場合，これらと別の名前をつけるのが適切であるかどうかについては疑問があります．Cのそれをモデルにしたのであれば，同様の省略記法を使用するほうが自然であるといえるでしょう．

ですから，原則は原則として，なるべく守るようにとしかいえません．ただし，自分のプログラムの中では，これらの記法は少なくとも統一されていることが必要です．統一されていれば，それがどんなルールであってもそれなりに美しく見えるものです．種々雑多なものが混じっていると往々にして汚いソースになります．

デバッグ

見落としと思いこみの産物を見つけるために

最後に，アセンブラのデバッグについて書いておきましょう．

机上デバッグがいちばんという人もいますが，筆者のような人間にはやはりcut&tryが性に合っています．もちろん，ソースをながめるというのは大切ですが，とれにくいバグというのは往々にして見落としと思い込みの産物ですから，それだけでは見つかりにくいのです．

デバッガを使用すると，このちょっとした思い違いを発見するのにたいへんに役に立つことが多いのです．ですから，筆者は積極的にデバッガを使用してもよいと思います．

98で使用することを前提とすると，デバッガとしてはCV (CodeView)，TD (TurboDebugger)，それと国産のPT (Partner) が考えられます．98環境では，CVは少し落ちるという話です．入手性にも多少難があると考えられ，実際にはTDかまたはPTを使うことになるでしょう．これらには，それぞれ癖がありますが，CVとPTは比較的操作性やコンセプトが似ており，TDはそれとは大きく異なっているという感じです．ですから，CVになれている人はPTは比較的容易に使用できると思いますが，TDを使うとちょっととまどってしまうと思います．逆にTDになれている人はPTやCVは使いにくいと感じるでしょう．

このほかに，さすがにdebugを使うことはないでしょうが，symdebを使う環境が考えられます．このクラスのデバッガは，とにかく「きびきび」した動きが期待できます．それにサイズ

ワンラインNEWS
YHPは建築設計事務所と共同で，約10万種類の住宅設計をシミュレーションできるデータベースシステムを構築した．同社のワークステーションで稼働する．

が小さいこともメリットです．そのかわり，シンボリックデバッグをする場合には，mapファイルに変換プログラムをかけてシンボルファイルを作り出すという必要があり，これはかなり手間がかかります．

symdebはサイズが小さいので以前はよく利用されました．TDやPTなどのデバッガは，普通に使用すると200KB以上ものメインメモリを使用します．ですからちょっと大きなプログラムをデバッグしようとするとメモリがたりなくなってしまいます．まあ，アセンブラプログラムの場合には，それほど大きなプログラムにはならないでしょうが，デバッガの本体部分をEMSやGRAM，ファイルなどにスワップアウトしてメモリ常駐量を削減するというオプションがついていたものもありますが，今度は速度が極端に落ちてしまいます．

これに対して，symdebだと3～40KB程度のオーバーヘッドが必要なだけですから，その分大きなプログラムにも使えたわけです．しかし，現在のウィンドウデバッガは286/386CPUで使用する場合には，プロテクトメモリに本体をロードできるようになっているのが普通です(98版のCVでは対応していない)．この機能を使用すれば，実際問題としてメインメモリの圧迫はほとんど無視できます．ですから，サイズの面でもsymdebの優位性は薄らいできています．使われるとしたら，むしろsearchなどのコマンドのスピードによって，でしょう．あと，プログラムにパッチを当てる場合にも，symdebのほうがやりやすいはずです．

アセンブラのデバッグであっても，高級言語と同様にウィンドウデバッガのほうが操作性がよいはずです．ブレークポイントの設定も容易にできますし，ステップ処理，レジスタや変数の参照などがやりやすくなります．

とくに，マクロなどを多用している場合には，ウィンドウタイプのデバッガのメリットは大きく出てきます．マクロを使用すると，ソースレベルで1行である記述が，複数の機械語に翻訳されることがひんぱんにあります．逆アセンブルリストを見ながらのトレースだと，ソースステップと実際の実行ステップとの間の対応がとれないので，かなり厄介です．その点ウィンドウデバッガなどでソースそのものを参照し，ソースレベルでステップやブレークポイントの設定を行えれば非常に楽になります．構造化マクロなどを利用しているときに，ソースレベルのデバッグと機械語モードでのデバッグとの作業を比べてみればよくわかることと思います．

ウィンドウタイプのデバッガだと，トレース操作が1キーでできるので非常に楽ですし，画面上で実行ポインタがリアルタイムに動くので制御の流れを追うことが容易です．この制御の流れを見ているだけで，なんとなくおかしい部分が判ってくることも少なくありません．また，レジスタウィンドウでつねにレジスタの値をチェックすることができます．symdebなどではrコマンドを実行しないとこれらがチェックできないので，かなり不便です．

以上のようにデバッガを使うとプログラムの実行状況がよく把握できます．ですから，簡単なアセンブラプログラムであっても，「つっかえてしまったとき」には積極的にデバッガを使って，どこでひっかかっているのかを早目に見つけ出すほうが無難です．

もちろん，ただ漫然と始めから終わりまでステップ実行すれば問題点が見つかる，というものでもありません．おのずと対象となるポイントの絞り込みの技術が必要になります．バグでいちばん厄介なのは，動作はしているようだがちょっと変だというものです．しかし，それよりはるかに多いのは，暴走してしまう，というものでしょう．とくにアセンブラでは，高級言語以上にスタック関係などの誤操作で暴走しやすいものです．暴走の場合，プログラムが正常に動いているかそうでないかを判断するのが比較的容易ですから，その分楽といえます．

暴走の場合，最初にしなければいけないのは暴走する環境の確定です．つねに暴走するなら問題はないのですが，動いたり動かなかったりする場合，何が問題なのかを調べる必要があります．パラメータを渡して動かすプログラムであれば，パラメータの値に依存するかもしれません．メモリ位置やスタックサイズに依存するかもしれません．そして，デバッガの上で走らせた場合でも，同様な条件で暴走すれば完璧です．

デバッガの上で問題が再現できれば，次には実際に問題を起こしているコードを探し出すことになります．プログラムを書き換えても大丈夫（同様な問題が再現する）であれば，画面出力やそのほかのプローブをほうり込んでどこまで実行が進んでいるかを調べることができます．あぶなそうなところがわかれば，そこにブレークポイントをおいて，そこから動作をチェックすることになります．特定条件でサブルーチンを呼んだときに暴走する場合，ループである回数になったら暴走する場合などは，その条件を満たすダミーの分岐分（ifブロック）を作成し，そのときにだけ実行されるnop命令か何かにブレークポイントを置けばよいことになります．また，直接int3の割り込み命令を書いてしまってもかまいません．int3を使う場合はもちろん，正式版では絶対にこのコードが残らないようにしないといけません．

このような動作確認が不可能なら，最後にはトレースでチェックするしかありません．トレースには，1命令ごとにハードウェアブレークして止まるステップと，命令を解釈して次の命令にブレークポイントをおくことで，call命令をスキップできるものとがあります．後者をステップと呼びましょう．最初にトップレベルでステップを実行します．きっと，どこかのサブルーチンの呼び出しを実行したところでハングアップするでしょう．リセットをかけてまたデバッガでプログラムを読み込み，さっきハングアップしたところまで実行を進めてから，今度はトレースをかけて1つ下のサブルーチンまで飛ばします．そして，同様にステップ命令でハングアップするまで実行させます．

このような動作を（リセットを交じえながら）実行することで，だんだんとサブルーチンの絞り込みが行えます．再帰的に呼ばれる関数など非常にたちの悪い関数では，簡単にいかない場合もありますが，たいていの場合ハングアップするまでの過程で1回しか通らないポイントがいくつかあるもの

ワンラインNEWS
ソフトバンクはインテルジャパンと，パソコンLAN用の関連機器について販売契約を結んだ．インテル製のLANで用いるプリンタの共有装置をソフトバンクが販売する．

です．そこにブレークポイントをおきながら実行を進めていくことで，下請けルーチンで特定の環境でだけ問題が発生する，というような場合でもトラップをかけることが可能になります．

このように複数のブレークポイントをたどっていくというような決まりきった操作をする場合には，デバッガのマクロ機能が役に立ちます．TDのマクロはいわゆる「キーボードマクロ」であって，かなり制限のきついものですが，それでもこの程度の処理には十分でしょう．PTの場合には条件判断まで使うことができるので，非常に柔軟なマクロ処理が可能です．

さて，そうしてだんだんと絞り込んでいくと，最後には「変だ」と思うポイントに行き当たります．暴走のたぐいの場合，問題点はかなり狭い範囲で解決可能なものです．たとえば，スタック操作の問題があってret命令で変なところに飛んでいってしまうような場合，まちがった操作をしているのはその関数の中に限られます．ジャンプ先ラベルがまちがっているような場合など，そのジャンプ命令をなおすだけです．

無限ループも暴走の中に含められることが多いですが，この場合もループの終了条件をチェックすることで問題点がはっきりすることがほとんどです．むずかしいのは，問題をもっているループを探し出すことだけなのです．無限ループの場合でも，問題になる部分は関数の中に限られる場合がほとんどでしょう．大域変数や引数にとった値が問題を起こしている場合も稀にあります．その場合には，その引数を作った部分が「正常に動作していない」というわけですからその部分をチェックしてくことになります．

暴走は，前記のように正常に動作しているかどうかを判断するのが容易で，しかもどこまで正常に動作しているかをチェックするもの簡単ですから，デバッグは非常に簡単な部類に入ります．問題は，プログラムが動作はするけれど期待しているものと違った動作をしている場合です．適切なパラメータを作成していない関数が見つかった場合なども同様です．

誤動作をする場合でも，どこが誤動作をしているのかを見つける必要があります．関数単位で考えると，誤動作は関数の外側に不適切な値を返しているわけです．これは，ファイルなどの外部環境に値を返す場合を除けば，返り値がおかしい場合と外的変数（場合によってはポインタを渡された変数）に不適切な値を設定している場合です．ですから，まず不適切な現象を起こす部分を見つけることになります．

実際には，現象からたどるほかに手はないですから，ともかくおかしい結果を返す部分を探します．そして，それが期待通りになっていない理由を探します．外的変数が期待通りの値になっていない場合は，その変数に値を設定する部分を探して，そのルーチンをチェックします．

場合によっては，インダイレクトアクセスのアドレス計算がまちがっている場合があるかもしれません．そして，本来は書いてはいけない部分に書き込みをしているのかもしれません．そのようなエラーは，ソースレベルで発見するのは困難です．変数のアクセスによってトラップをかけられれば，異常な値を設定している部分を検出することができる場合があります．PTはハードウェアで特定アドレスレンジのアクセスを検出できましたが，最近のシステムでは拡張バスにメモリアクセスのサイクルが出ないのでうまく働きません．そのかわり（これはTDでも可能ですが）386のデバッグレジスタを使用することによって，特定アドレスのアクセス検出が可能です．

また，386の仮想86モードで動かすことを許容されるなら，特定のI/Oアドレスにアクセスすることによってもトラップをかけることが可能です．デバッグレジスタは1～4バイトのレンジにしかトラップをかけることができませんから，特定の配列全体などに不正アクセスしている，というような状況でトラップをかけることは困難です．TDではページフォルトを利用してレンジアクセスの検出ができるようですが，これは下手をするととても遅くなります．

このようなハードウェアのサポートで，おかしい結果を出している部分がトラップできればしめたものです．結局問題になっているのはソフトウェアのアルゴリズムなのですから，そこまでくればアルゴリズムを検討すれば問題は解消できるでしょう．そのルーチン内でトレースをかけて，制御の流れやデータの内容の変遷をチェックできるとさらに有利かもしれません．

いずれにしろ，デバッグをするなら386CPUを積んだマシンを使用することです．メモリ使用量の削減，デバッグレジスタの使用などはデバッグ時に大きなメリットとなります．ただし，この場合はプロテクトモードに移行する必要があることから，一般に仮想86タイプのドライバとの共存はできません．仮想EMSを使っているものや，UMBに常駐ものをロードしているようなconfig.sys／autoexec.batを利用している場合には，デバッグ時にはこれらを書き換える必要があります．デバッグ専用の書き換え環境と，通常の実行環境とをわけて保存しておくこ

となるでしょう．

常駐プログラムのデバッグ

常駐オプションをもったデバッガが不可欠

同じプログラムでも，メモリに常駐するもの，たとえばTSRやデバイスドライバのプログラムはデバッグがより困難です．とくにデバイスドライバやDOS，DOSから使用されるBIOSなどをリプレースするTSRについては困難の度合いが高まります．これらのTSR／デバイスドライバが活性化されるのは，DOSから呼び出しがあった場合です．この場合，デバッガでブレークポイントなどを設定したとしても，デバッガ自身がDOSの呼び出しをいっさい使用できない状況になっています．たとえばsymdebでトラップをかけた場合，トラップがかかったというメッセージがDOS経由で出されるために，そのあとの実行の継続はいっさい不可能になります．

通常のTSRであっても，これらを活性化させるイベントがプログラムによって明示的にひき起こされる場合でないと，デバッガ上でプログラムが走っている間にイベントが発生するとは限らないので厄介なことになります．これが可能だったとしても，読み込んでいるプログラムはデバッグ対象のプログラムではないのですから(！)シンボリックデバッグは不可能になります．

ですから，これらのデバッグ動作にはPT／TDなどの「常駐オプション」をもったデバッガが不可欠になります．このオプションをつけると，デバッガ自体がメモリに常駐します．これによって，いかなる状況にあってもデバッグ割り込みがかかれば，デバッガの制御下に制御を移すことができます．もちろん，DOSの入力待ち状態からデバッガが起動されることも考えなければいけません．このため，デバッガのキー入力はBIOSを使用，画面出力は直接VRAM書き込みを使用し，DOSを経由しないで入出力をします．ですから，画面を直接触るプログラムやキーボードをトラップするようなプログラムでないかぎり，デバッガの実行は妨げられないことになります．

こうしてデバイスドライバの内部，あるいはDOSの内部さえもトレースとして実行状況をチェックすることが可能となります．また，別にシンボル情報を読み込み保存することによって，常駐しているプログラムやドライバに対してもシンボリックなデバッグが可能です．

デバイスドライバなどは，外界に働きかける方法が非常に制限されます．たとえば普通のプログラムでは，内部状況を（DOS経由で）出力することによって，比較的情報をたくさん得ることができます．しかし，デバイスドライバではこのような出力が許されないので，ハングアップしたとしても動作状況がどのようになっているのかわかりません．

そこで，デバッガの重要性が増してくるのです．デバイスドライバや常駐プログラムなどを作成する場合には，これらのデバッガを備えて使いこなす必要があると思います．

デバイスドライバの場合，デバッグ中はconfig. sysに記述することは絶対に避け，adddevなどで付加するようにします．このようにしないと，init時にハングアップするようなバグを作ってしまうと，あとでシステムが立ち上がらなくなります．最悪の場合，フロッピーディスクからたち上げなければいけなくなってしまいます．adddevならこのような問題は回避できます．

しかし，なんらかの理由でadddevできないドライバの場合はどうしたらよいでしょう．ブロックデバイスドライバなどがこれにあたります．その場合は，ドライバをフロッピーディスクにおき，config. sysではフロッピーディスクのドライバを読みにいくようにするとよいのです．もしもfatalなバグが出た場合には，フロッピーディスクを抜いておけばシステムは問題なく立ち上がるようになります．

ワンラインNEWS
NEOPTはワークステーションに接続し，スーパーコンピュータ並の処理速度を実現する画像処理ボード「Maxvideo20」の輸入・販売を開始した．

情報処理技術者試験
第2種合格！トレーニング
第7回　プログラム流れ図

静岡情報ビジネス専門学校　望月照夫・沖山貴裕

出題の傾向と対策

プログラム流れ図の問題は，午前の部，午後の部ともに出題されています（表1，2）．

午前の部では流れ図全般の論理を問うもの，配列処理（合計や整列等）に関するもの，基礎的な数値計算に関するもの等が出題されています．解答形式は多岐選択式です．

午後の部では毎回3問が出題され，そのうち1問を選択すればよいことになっています．3問の内訳は，事務処理に関するもの1問，数値計算に関するもの1問，その他（配列，文字列，数値計算等）1問となっています．解答形式は多岐選択式と記述式の両方が出題されています．午後の部の流れ図の配点は15点といわれています．比較的平易なものが多いので，ぜひ満点を取るようにしてください．

流れ図は事務処理から数値計算まで出題範囲が広いので，とくに午後の問題対策として事務処理または数値計算などの得意分野を1つ作っておく必要があります．さらに言語と関連づけてCOBOLなら事務処理，FORTRANなら数値計算というように選択するのが早道です．2種受験者には数学を苦手とする人も多いようですが，事務処理の流れ図とCOBOLを選択すれば十分に合格は可能です．

配列処理に関しては，どの分野を選択するにしてもマスターしておく必要があります．配列に関しては整列を十分理解しておくこと，選択法（最小値法），隣接交換法，挿入法等種々のバリエーションで出題されるのでそれぞれの処理の違いをよく理解しておくことがたいせつです．

事務処理に関しては，順ファイルのコントロールブレイクとマッチングが基本ですが最近はレコード内容の編集，印字の問題が多く出題されています．

数値計算に関しては，式の解法，素因数分解，二項定理，数列，求積とあらゆる分野から出題されています．内容は中学校の数学から大学の初等数学までの範囲です．しかし，問題が難しいほど処理の流れが問題文の中に説明されているので，問題文をよく読めば解答できることもあります．

配列処理

配列に関する問題のうち出題頻度が高いのは，配列要素の整列（内部SORT）についての問題です．内部SORTの種々の方法のうち，これまでに選択法，隣接交換法，単純挿入法が出題されています．

■選択法

選択法は別名最小値（最大値）法と呼ばれ，図1のように，配列の先頭の

●表1　午前の部出題一覧

平成3年　春	問4	決定表
平成2年　春	問4 問5 問6	条件判定 配列の整列 配列の合計
平成元年　春	問9	代数計算
昭和63年　秋	問8	配列の整列
昭和63年　春	問5	全般的な論理
昭和62年　秋	問6	配列の合計

●表2　午後の部出題一覧

	事務処理	数値計算	その他の分野
平成3年　春	レコードの印字	関数と配列	文字列操作
平成2年　秋	配列を用いた印字	二項係数	文字列操作
平成2年　春	得点の降順印字	仮分数の計算， 積分による求積	
平成元年　秋	ファイル処理	多項式	データ構造
平成元年　春	コントロールブレイク	数値の配列処理	文字列操作
昭和63年　秋	ファイルの併合	多項式	運賃計算
昭和63年　春	配列を用いた印字	漸化式	配列の整列
昭和62年　秋	ファイルの照合		最大値・最小値・平均値， 最頻値と度数
昭和62年　春	レコードの順番検査	素因数分解	ヒストグラムの作成
昭和61年　秋	ファイルの更新	関数とグラフ	文字列操作

要素と２番目以降の要素を順次比較し，より小さい要素（昇順の場合）を先頭に格納する方法です．最後の要素との比較が終ると，最小値が先頭にきます．２番目以降の要素について同様の処理を繰り返していけば，整列が完了します．

■隣接交換法

隣接交換法による昇順の整列では，**図２**のように配列の先頭から順に隣の要素を比較し，値の大きい方を後ろに格納します．配列の終りの要素まで比較が終了すると配列の最後には最大値が格納されます．同様に残りの要素について処理を繰り返していけば，整列が完了します．

■挿入法

挿入法は**図３**のように，ファイルからデータを読みながら整列を行う方法です．データを配列の要素と比較し，挿入すべき位置を決定します．次に，その位置から後ろの要素を後ろから順次１つずつ後ろにずらして，新しいデータを格納する場所を空けてからデータを格納します．

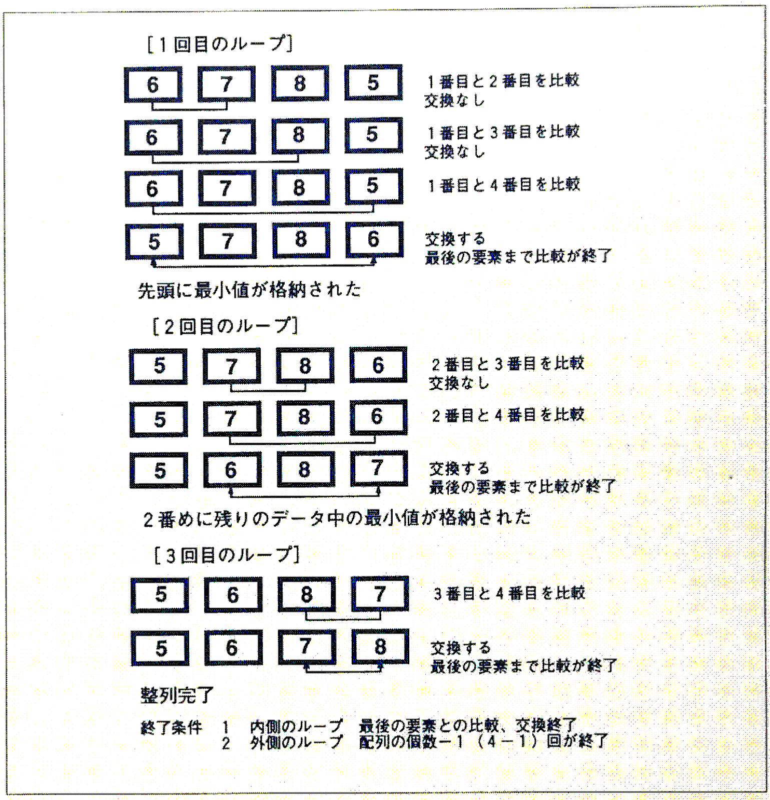
EXAMPLE 1

整列に関する流れ図中のａ～ｄに入れるべき適切な字句を，解答群の中から選べ．

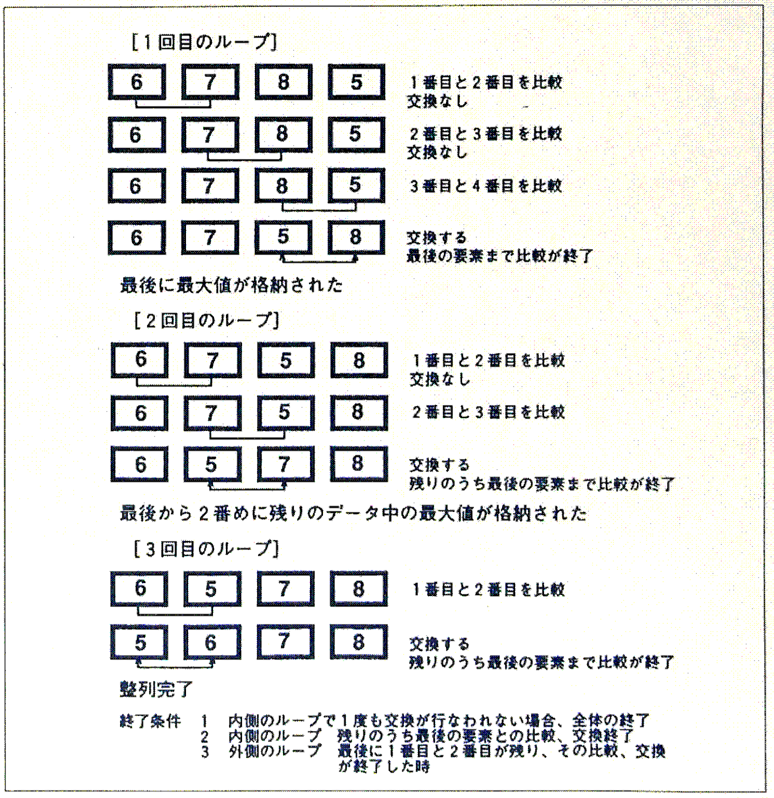
［流れ図１の説明］

配列Ｈ(Ｉ)(Ｉ＝1,2,3,……,Ｎ)に入っている数値データを昇順に並べ替える処理である．並べ替えは，“配列の後ろから順に隣り合う要素同士を比較し，大小関係の逆転がある場合には値の交換を行う”という走査の繰り返し処理で行われている．図は最初の走査の様子を，Ｎ＝5の場合を例として示したものである．

●図１　選択法（最小値法）



●図２　隣接交換法



H(1)	H(2)	H(3)	H(4)	H(5)	交換
35	50	10	99 ＞	43	する
35	50	10 ＜	43	99	しない
35	50 ＞	10	43	99	する
35 ＞	10	50	43	99	する
10	35	50	43	99	

図に示されるように，最初の走査によって最も値の小さい要素がH(1)に入る．したがって，2回目の走査はH(N)からH(2)まででよいことがわかる．このように，範囲を縮小しながら走査をN-1回繰り返すと求める結果が得られる．ただし，途中で1度も交換が起こらない走査があれば，すでに並べ替えは終わっていることになる．なお，流れ図中のXおよびWはワークエリアである．

［解答群］

ア．0→X　　イ．0→W

ウ．1→X　　エ．1→W

オ．H(J)→W　カ．H(J)→H(J-1)

キ．H(J-1)→W

ク．H(J-1)→H(J)

（第2種　平成2年春　午前　問5）

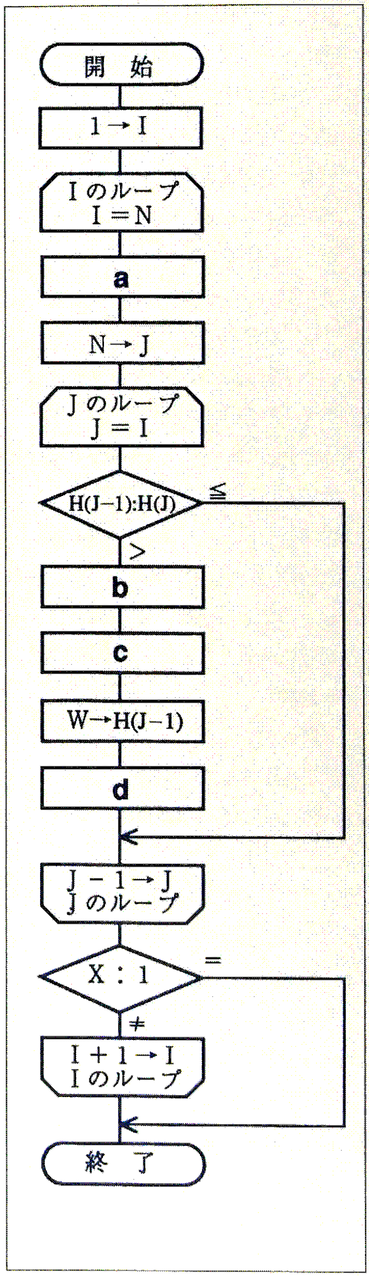
解説

交換法の問題です．ただし比較を配列の末尾から行い，小さい方の要素を順次前の要素に移動し，最小値を配列の先頭に格納します．Iは走査の回数，Jは配列H(I)の添字です．外側のループは走査の回数を制御し，内側のループで配列要素の比較と交換を行っています．また，Iは走査の回数（初期値1，増分1，終了値NでN-1回）を数えるだけでなく，内側ループの終了条件でもあります（2回目の走査，I＝2ではH(N)からH(2)までを比較すればよい）．H(J−1)＞H(J)の時の入れ替え処理は，Wを介して行われます．また，Xは入れ替えが行われたか否かの判定に用いられていることに注目してください．

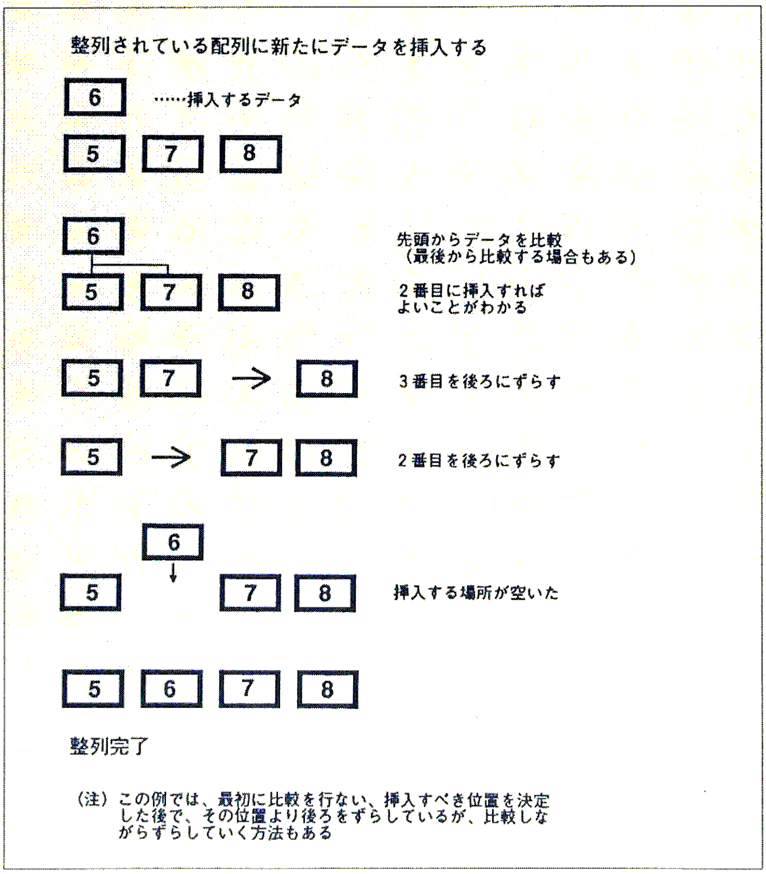
解答

a－ウ，b－オ，c－ク，d－ア

●流れ図1



●図3　挿入法



事務処理

事務処理の分野からは，レコードの編集印字の問題が数多く出題されています．また，グループ合計等については，次回のCOBOLであらためて取り上げます．

EXAMPLE 2

次の流れ図の説明および流れ図を読んで，設問に答えよ．

[流れ図2の説明]

所得ファイルを読み込んで，年齢層別に所得額の最高額を求め，印字する．印字様式は，図4のとおりである．

(1)　所得ファイルの各レコードには，所得額と年齢が格納されている．

(2)　最高所得額を求めるために，TABという1次元の配列を用いる．TABの要素は6つである．

(3)　各年齢層に該当するレコードは，必ず1件以上存在する．

(4)　除算において，小数点以下は切り捨てられる．

[設問]

流れ図中の(a)～(c)に入れるべき正しい答えを解答群の中から選び，その記号を記入せよ．

[解答群]

ア．所得額：TAB(K)

イ．TAB(K)：所得額

ウ．所得額：0

エ．TAB(K)：0

オ．K：6

カ．年齢÷10→K

キ．年齢－10→K

ク．年齢÷10→年齢

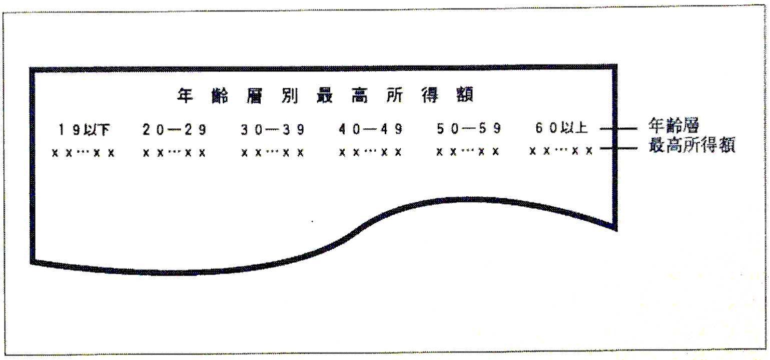
ケ．6→K

（第2種　平成2年秋　午後　問3）

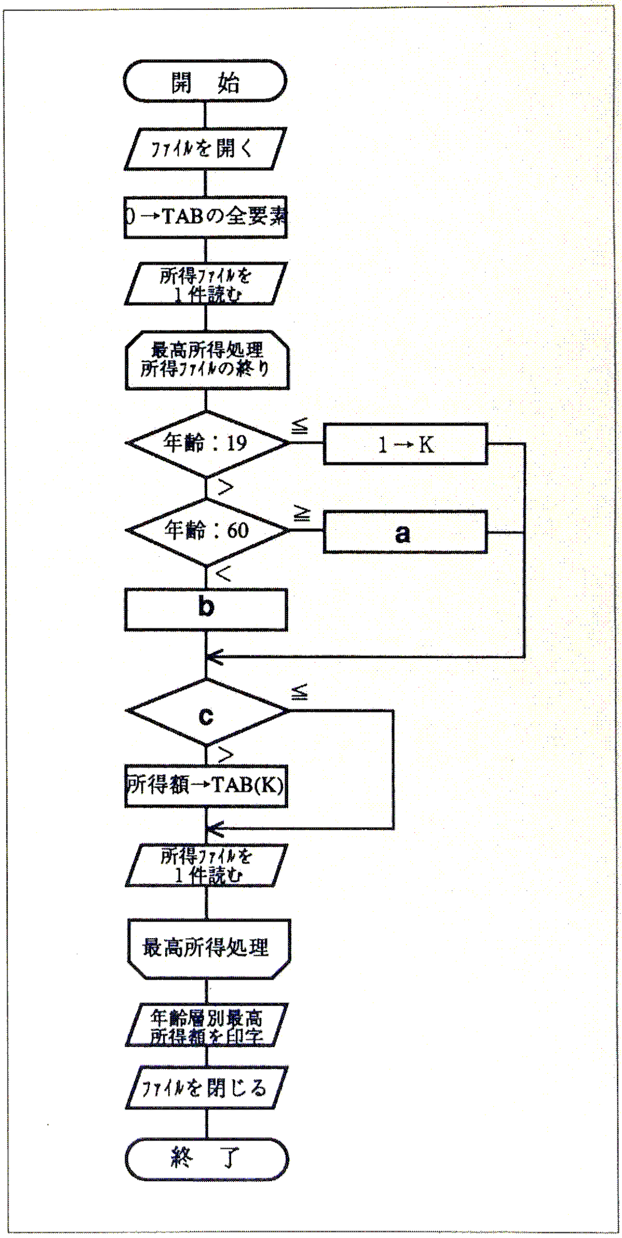
解　説

レコードの編集印字の問題は，午後の事務処理の問題ではもっとも多く出題されています．その内容は，レコー

●図4　年齢層別最高所得額の印字様式



●流れ図2



ドの特定の項目を抽出し，一定の形式に編集しなおすもの，またテーブルを用いて集計をしてから印字するもの（ヒストグラムもこれに含まれる）などがあります．

例題のようにテーブルを用いる場合には，レコード中の特定の項目の値を，格納すべきテーブルの要素を決める添字に使うことが多いのです．この例題では集計用テーブル（TAB）の要素は6個であり，年齢層と対応づけられています．流れ図中で年齢≦19の場合に添字Kを1にセットしていることから，(a)の処理では年齢≧60の場合はKを6にすればよいことがわかります．また(b)では，19＜年齢＜60の場合に，20～29の場合K＝2，30～39の場合K＝3のように，その年齢層に応じた添字の値を計算すればよいことにも気がつくでしょう．また(c)では，いま読み込んだ所得額をテーブルに格納するか否かの判断（その金額がいままでの最大値の場合格納する）を行っていることがわかります．

解　答

a－ケ，b－カ，c－ア

数値計算

数学系の問題はあらゆる分野から出題されているので，限定するのは難しいのですが，ここでは，一見難しそうに見えるけれども，問題をよく読めば解答できるものを取り上げてみました．

EXAMPLE 3

次の流れ図中の(a)～(c)に入れるべき適当な答えを図にならって記入し，流れ図を完成せよ．

［流れ図の説明］

x-y 平面上の n 個の点 (x_1, y_1)，(x_2, y_2)，…，(x_n, y_n) について，その回帰直線 $y=e+fx$ の係数 e と f を次の式で求める．

$$f=\frac{(x_1-\bar{x})(y_1-\bar{y})+(x_2-\bar{x})(y_2-\bar{y})+\cdots+(x_n-\bar{x})(y_n-\bar{y})}{(x_1-\bar{x})^2+(x_2-\bar{x})^2+\cdots+(x_n-\bar{x})^2}$$

$$e=\bar{y}-f\bar{x}$$

ここで，

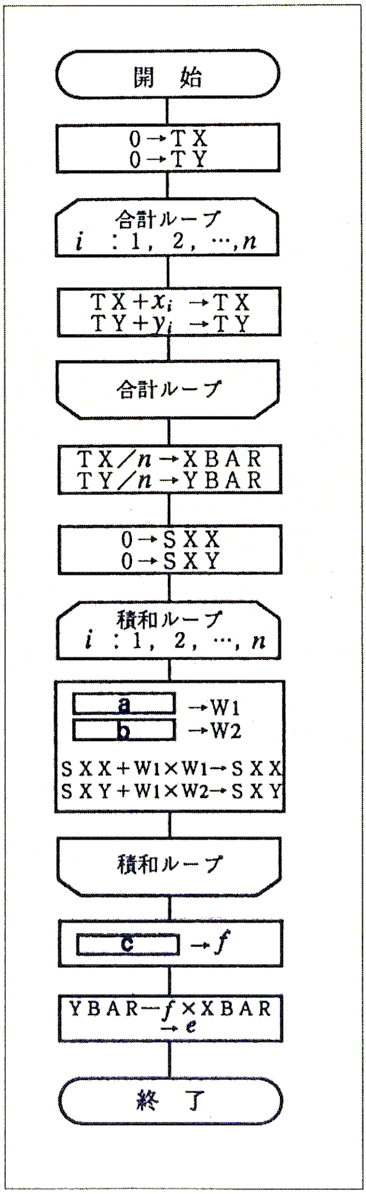
$$\bar{x}=(x_1+x_2+\cdots+x_n)/n$$

$$\bar{y}=(y_1+y_2+\cdots+y_n)/n$$

である．また n>1であり，f の分母は0にならないものとする．

（第2種　昭和63年秋　午後　問2）

●流れ図 3



解　説

回帰直線という言葉を見ただけでやめてしまう人も多いと思いますが，式のとおりに順番に計算していけば，容易に解ける問題です．最初のループでは，x_1・・x_nの総和TX，y_1・・y_nの総和TYを求めています．それを n で割ったXBAR，YBARはそれぞれ $\bar{x}$，$\bar{y}$ を指していることがわかります．次の積和ループでは，f を求めるための分母，分子を求めています．式より，SXX＋W1×W1は分母，SXY＋W1×W2は分子ということがわかります．したがって，

（a）　xi－XBAR

（b）　yi－YBAR　となります．

このループを抜けるとSXXには分母がSXYには分子が入っているので，

（c）　SXY／SXX　となります．

解　答

a．xi－XBAR

b．yi－YBAR

c．SXY／SXX

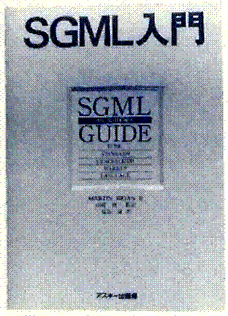
Book Review

■中村正三郎

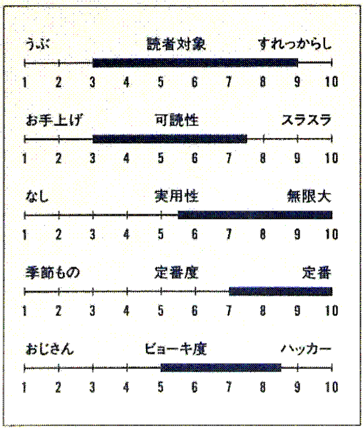
多様な文書処理を可能とするSGMLの概念，利点をこと細かく解説

■SGML入門

MARTIN BRYAN 著
山崎俊一監訳　福島誠訳
アスキー出版局　4,200円



■評価チャート



本書は，標準汎用マークアップ言語SGML（Standard Generalized Markup Language）とは何かを，豊富な例を交えて深く解説した本である．

欧米では，SGMLはすでに実用化段階に入っており，電子文書を大量に扱う公的機関や大企業での導入が始まっている．

SGMLは，文書の構造をタグというもので表現する．これをマークアップという．タグ付けによって構造を明確にした文書は，文書交換，電子出版，文書データベースなど，さまざまな応用に適用しやすい．タグを手がかりに，多様な処理を文書に施すことが可能になるからである．一例を挙げれば，ECでは公的会議の議事録はSGML文書として作られているそうである．これは，同一内容の議事録をいくつものEC加盟各国の言語で記録に残さねばならず，そのデータベースにSGMLで登録しておくことによって，必要な部分のみのアクセスを可能にする試みであるという．議事録に，フランス語部分であることを示すタグ，ドイツ語部分であることを示すタグ，といった具合にタグ付けしておけば，各国からのアクセスに対して，当該国の言葉で書かれた部分のみを容易に抽出できる．SGMLを使えば，このような応用も考えられ，出版業界，印刷業界，データベース業界も巻き込んで注目を集めている．

文書を構造にしたがってタグ付けする考え自体は目新しいアイデアではないが，SGMLではタグそのもの，その使い方も可変であり，特定の出力媒体，処理系などに縛られず，非常に一般化された形でマークアップが可能になっている．これが，generalizedマークアップ言語という名称の所以になっている．SGMLは，そのようなマークアップ言語の国際標準である．

規格書は，ISO 8879である．SGMLが先に述べたように，マークアップの方法そのものも記述できる，いわばメタ言語であるゆえに，この規格書は難解な規格書として知られている．付録（Annex）から読まなければ，なかなか概念が理解できない．そういう代物である．現実にSGMLに接する人たちが，そのような規格書を読んで理解するのは，ほぼ不可能である．

そこで，本書の登場となる．著者は規格制定の中心人物である．SGMLの概念，その利点などをこと細かに例を交えて解説している．

以前，この欄で紹介したオブジェクト指向入門もそうだったが，この点，本書の入門というタイトルは，いささか的はずれである．ともにアスキーからの出版である．アスキーの担当者には，安易に入門をタイトルにつけるのではなく，もう少し工夫するよう望みたい．

約380ページに及ぶ分厚い本であり，SGMLの用語が頻出するため，読み進めるには困難があるだろう．しかし，SGMLを理解しようする読者は，その労苦がきっと報われるはずだ．

わが国でも，JIS化の作業が行われており，関心が高まっている．コンピュータ業界のみならず，出版業界，印刷業界，データベース業界の方々も一読されるとよいと思う．

ワンラインNEWS
トーメンエレクトロニクスは英IX社のターミナルエミュレータソフト「X・deskterm」の輸入，販売を開始した．Xウィンドウ端末用のCUIソフトにGUI機能を付加する．

なお，付け加えておけば，本書自身もSGMLによってマークアップされた文書を印刷したとのことである．その作業の概要が最後に掲載されている．

続々とUNIX関連図書の訳本が登場

古田島義和（神田・書泉グランデ）

ますます快調なUNIXに熱い視線が向けられています．年々UNIX関連図書が増える中で，先月紹介した丸善刊の「UNIX4.3BSDの設計と実装」が予想通り月刊ベスト10の第1位に輝きました．バークレー版UNIXはAT&TのUNIX System Vを基にカリフォルニア大学バークレー校で開発されたOSで，いま注目のネットワークプロトコルTCP/IPもこのOSの4.2BSDから実装されました（BSDはBerkeley Software Distributionの略です）．バークレー校では，1989年夏に“386BSD”のプロジェクトが発足．インテル80386を搭載したマシン用のOSが動きはじめています．これに関する図書はまだ出版されていませんが，参考になるものといえば，Addison Wesley（以下AW）「The Design & Implementation of the 4.3BSD Unix Operating System」とSylex「Programming the 80386」が最適だと思います．

UNIXを細分化した解説書がO'Reillyから次々と出版されています．これらの本は啓学出版，アスキーなどから訳本として出版され，ベストセラーになっています．UNIXプログラミング関連でもっとも売れているのは，L. Wall&R. Sbwavtz「Programming perl」，D. Dougberty「Sed&awk」，同じ著者で「Lex&Yacc」，G. R. Gircys「Understanding and Using COFF」，新刊がD. Dougberty「POSIX programmer's Guide」，J. Bloomer「Power programming with RPC」，S. Oualline「Practical C programming」．啓学出版から訳本として，I. F. Darwin「Checking C programs with lint」とS. Talbott「Managing project with make」，第3弾としてJ. Strang「Programming with C users」が出ています．アスキーから出たD. Curry「Using C on the UNIX System」，UNIXシステム管理分野でM. Loukides「System performance Tuning」が今回の和書ベスト10の第2位，第3位を占めています．ほかにシステム管理分野で売れているものに，J. Strang，L. Mui，T. O'Reilly「Termcap & Terminfo 3/E」，新刊でH. Stern「Managing NFS and NIS」が評判を呼んでいます．UNIXセキュリティ分野では，洋書・和書とも最新のものがなく，よく売れているのが，S. Gerfinkel&G. Spafford「Practical UNIX Security」です．UNIXのブームにのって彗星のごとく現れたT. O'Reillyの次の1冊が楽しみです．

月刊ベストの中で第22位にマグロウヒル「OLTPシステム」があります．これは，ネットワークシステム高度化を支えるオンライン・トランザクション処理に関する本で，UNIX採用によるオープンシステム化，ネットワークホストに加え分散処理などの拡大にともなって注目の1点になりました．ほかに第27位に啓学出版「インサイドGNU Cコンパイラ」はFree Software Foundation「Internals of GNU C」の訳本で，発売の週第4位で上昇中．来月のベスト10上位入りは確実といわれるほどの人気です．このGNUはR. M. Stallman氏が開発したもので解説書としては，共立bit別冊「GNU Emacsマニュアル」に次ぐものです．GNU Cコンパイラを理解するのに最適な1冊です．洋書のベスト10を見てもわかるように，コンパイラがいかにプログラマにとっての関心事であるかがうかがえます．

■和書&洋書ベスト10

（書泉グランデ調べ：1991年7月1日～7月31日）

	和書	出版社		洋書	出版社
1	UNIX4.3BSDの設計と実装	丸善	1	Compilers	Addison Wesley
2	UNIXシステムチューニング	アスキー	2	Programming Perl	O'Reilly
3	UNIX Cプログラミング	アスキー	3	Undocumented DOS	Addison Wesley
4	電脳騒乱節Vol.2	技術評論社	4	ResEdit Reference 2/E	Addison Wesley
5	電脳騒乱節Vol.1	技術評論社	5	Programming Windows 2/E	Microsoft Press
6	コンピュータ・ネットワーク	岩波書店	6	Simple Book : An Introduction to Management of TCP/IP	Prentice Hall
7	C言語による最新アルゴリズム事典	技術評論社	7	ResEdit Complete	Addison Wesley
8	日本語入力読本	翔泳社	8	System Performance Tuning	O'Reilly
9	マルチメディアへの挑戦	ソフトバンク	9	UNIX System v Release 4series	Prentice Hall
10	UNIXユーティリティライブラリmake	啓学出版	10	MacScheme : Manual and Software	M.I.T.

ワンラインNEWS
CSKは電子手帳を使ってデータをやりとりする無線データ通信システムを開発した．シャープの「DB-Z」に対応し，営業マン向けにレンタルを開始する．

◆◆ 佐々木哲のニューヨーク ◆◆

N.Y. システム通信

混沌の世界

短いサイクルのコンピュータの世界に限らず，世界は再編成に揺れ動いています．この原稿を書いている今，ソ連のゴルバチョフ大統領の失脚が伝えられました．旧ソ連・東欧ブロックは，秩序を取り戻すのにどれくらいかかるのでしょうか．

先月のBorlandによるAshton-Tateの吸収合併発表に続いて，Novellによる名門Digital Research Corp.の買収です．ユーザーは，OS/2の将来について，IBMとMicrosoftの間にはさまれ困惑しています．IBMによる，OS/2へのBorland C++，Watcom C，Fortranの採用など，Microsoftに背を向けるニュースばかりやけに目につきます．一方IBMは，大幅減益もいいところ．

それでなくても，アメリカは景気の後退が続いています．ユーザーの購買意欲は減る一方．それゆえ，今年前半は各ハードウェアメーカーが驚くほどの価格引き下げを行いました．そのうえでの業界の進路の混乱は，ますますユーザーの新規購入を控えさせるだけです．

IBM PC 10周年

この8月で，IBM PCが発売されてからもう10年になりました．Apple II，TRS-80からは14年です．肝心のIBMが振るわないため，なんら目だった行事も行われませんでした．PC/ATが発売された1984年には，アメリカの全パソコン市場で60%以上ものシェアを誇ったIBMも，今では35%ほどです．Appleに一時はトップを抜かれ，残りは大小さまざまのIBM互換機メーカーと分けあうありさまです．IBM PC互換機，DOSは年老いたとはいえ，まだまだ寿命のつきるのは先のことです．混乱と困惑の中の10周年です．さてまた10年後にはどうなっていることやら．

好調が続くDOS 5

かつてないほど，ベータテストを広く行ったMS-DOS 5ですが，その効果は十分あったようです．基本OSのバージョンアップにしては，驚くほどの数のユーザーがトラブルもなく移行しています．発売後数か月で，100万コピー以上出荷されています．MS-Windows 3.1のベータ版が配付される8月の開発者向けのコンファレンスも，1,800人もの開発者が参加したそうです．

この記事が出るのは10月号，秋のComdexのシーズンです．そして日本では待望の日本版のDOS 5がIBM系機種をはじめ，各社から発売されることでしょう．

プログラマの方も，ぜひこのDOS 5の正式なReference manual*1を入手することをおすすめします．DOS 5新規の機能のほかに，ありがたいことに，今まで明記されていなかったファンクションも多数「公認」されました．PC-DOS版はIBMより，またMS-DOS版のものはMicrosoft Pressより発売されています．

なお，以前にも触れたRalf BrownのInterrupt listも，6月にDOS 5対応版がでています．各バージョンでのファンクションの違いを把握するにはとても便利です．

MS-Windows 3のコツ

これだけ設定しにくいシステムが広まるのも驚くべきことです．設定条件の記録ファイルがsystem.ini，win.iniと無数にある．詳しい人でも一度トラブルが出ると解決に苦労します．ましてや，そのうえにインストールされるプログラムが，それぞれ勝手にこの設定を変えるものだから大変です．

ここらあたりの調整のコツをつかむには，MS-Windows 3のマニュアルは不十分です．Bantamより非公開のキーワードの解説を含めて，MS-Windows 3を120%使うための解説書*2が発売されています．600ページをこえる分量に，オリジナルのグラフィックバッチ言語，Command Post，IconDrawなどが入ったディスクが付いてきます．解説はPIFを含めたWindowsの設定，メモリ管理，ネットワークでの使用まで広範囲に渡っています．

アメリカでは，マイクロソフトより直接Windows Resource kitという補足マニュアルとディスクを15ドルほどで購入できるのですが，それよりも良質の書籍です．

ワンラインNEWS
日本IBMとSRAは，RISC型ワークステーション「RS/6000」などを販売する新会社「AIT」を共同出資で設立した．両社からの出向社員50人でスタートする．

フォーム処理プログラムの苦戦

一昨年あたりから流行した，フォーム処理プログラムが意外なことに苦戦しています．

業務で欠かせないフォーム類，いままではタイプライタ，あるいは手書きで前もって印刷された帳票にデータを記入していたものを，データエントリを含めDTP的に全てをコンピュータ内で処理し，高品位のプリンタで印字しようというものです．100ドルを切る低価格のものから1万ドルをこえる専用機のシステムまで，多数のシステムが発表されました．

上位のシステムでは，WYSIWYGでのフォームのデザイン，データエントリを提供し，さらには，多くのアプリケーションからのデータ読み込みなど，盛りだくさんの機能が組み込まれました．

ところが最近の報道によると，全体で業界の予想の8分の1程度しか販売が伸びなかったそうです．低価格帯で著名なFormWorx Corp.を含めて，他社に吸収合併されるところまでいくつもでました．

この原因は，ユーザーの本当のニーズをつかみ損ねたところにあるようです．ユーザーは，簡単にデータをエントリできるソフトを求めていたのに，データベース的な機能を充実してしまった．また多くの潜在ユーザーは，高機能化したワードプロセッサ，スプレッドシートで用を足してしまったということだそうです．ユーザーのニーズを知ることの重要さを教えてくれます．

ところで，日本ではこの手のソフトはほとんど見あたりません．これは上の理由にくわえて，ひとつにはコンピュータアプリケーションの使い方全体の未熟さもあるでしょうが，日本の社会に密着した問題があると聞きます．帳票を一部簡略化しコンピュータ化すれば大幅な手間が省けるところを，細部にこだわり，複雑で処理し切れないものを要求する．とくに，お役所に提出するような書式は，書体に渡るまで細かく指定されているため，現在の技術では処理しきれないというようなものもあるようです．

Clip artならぬClip music

Macintosh，グラフィックソフトの隆盛に伴い，無数のClip artが提供されています．ユーザーは著作権を気にすることなく，自分の作成するグラフィック，文書の中にイラストなどを張り付けて使用できます．

音声も含めたマルチメディアの時代にあわせ，Clip artならぬClip musicのCD-ROMが発表されました．Passport Designs Inc.からのMacintoshおよびMicrosoftのMultimedia PC用の製品です．

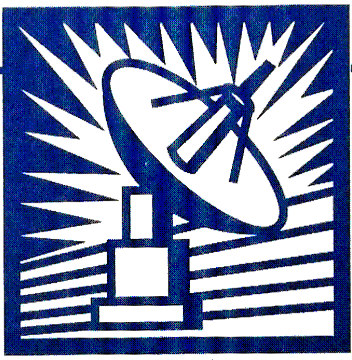
Media Maker*3は，各分野のマルチメディアプレゼンテーションで使える曲が120曲，MIDIだけでなく，8ビット，16ビットのデジタルオーディオのフォーマットで記録されています．同社のソフトウェアで編集できます．

ほかにも，Media samplerという20曲のみのCD-ROMも発売されます．

＊1 Microsoft MS-DOS Programmer's Reference, $24.95
Microsoft press, ISBN 1-55615-329-5
＊2 Windows 3 Power Tools Book/Disk $49.95
Geoffrey LeBlond, Bantam, ISBN 0-55335-298-9
＊3 Media Maker sampler $99
Media Music $395
Passport Designs Inc., 625 Miramontes St., Half Moon Bay, CA 94019
phone (415) 726-2254

Illustration:Masami Nozue

Illustration : NOBUKI FUKUHARA



西川和久の
希望的観測

キーボードの反乱!! どのキーボードが日本語入力に適しているか!?

1 OADGでキーボードタイプ追加 J-3100, AX, US101もサポート

先日, OADGからキーボードに関する記者発表があった. 内容としては, DOS/Vでサポートするキーボードタイプの追加である. 今までは, 日本IBMが対応していた1/2/3型鍵盤と, OADGキーボードと呼ばれるA01鍵盤だけがDOS/Vでサポートされていた. OADGに東芝や各AXメーカーが加わったことで, キーボードのサポートがどうなるのか, その行方が注目されていた. 「絶対統一されるわけがない」と思っていた私の予想は見事に的中したようだ. 「J-3100」「AX」「US 101」この3つのタイプが, 新たにDOS/Vでサポートされるキーボードタイプに追加された. 各メーカーがDOS/Vに対応するには, キーボード配置が一番問題だと思っていたが, これで一見落着である.

また, この秋米国でDOS/Vを出荷する予定もあるため, 拡張101キーボードにも正式に対応した. 現在, DOS/V版の日本語FEPでは, VJE-βが唯一拡張101キーボードにも対応しており, /IPS2オプションを付けると[Ctrl] + [Space] で漢字入力状態になる. ほかの日本語FEPでもこのような対応をするのであろう.

今まで, PC互換機でDOS/Vを使う場合, キーボードでいろいろ不便があった. オンラインで流通しているKEYBOARD. SYSやSIMKEY. EXEを使う必要があるのだ. デスクトップなら日本語キーボードと取り替えるという荒技が使えるが, ラップトップではそうもいかない. しかし, これはJ-3100に対応したことによってクリアできそうだ. J-3100のキーボードは, 米国の86キーボードを基本としているため, ラップトップタイプのPC互換機でもキートップを張り替えるだけで日本語対応可能である. カナ入力時に, 右 [Shift] の位置にある「ロ」を許せるかどうかは別問題であるが…….

2 DOS/Vパワーフォーラムもキーボードで終始 慣れたものがいちばん使いやすいのだが

ちょっと話は古くなるが, 5月に行われたDOS/Vパワーフォーラムのイベントのひとつに, DOS/Vについてユーザー側の意見を述べるパネルディスカッションがあった. 当初, 「VGA」「プリンタ」「キーボード」「PC互換機」など, さまざまな意見交換をする予定であったが, 蓋を開けて見ると2時間近く行われたディスカッションの内容は, キーボードに始まりキーボードだけで終わってしまった. 「ASCII配置のほうがいい」「[ESC], [Ctrl]の位置が嫌いだ」「しかしJIS配置のほうが……」など, 堂々めぐりである. これほどキーボードに対していろいろ思い入れがあるとは考えてもみなかった.

確かにコンピュータを使う時, 一番よく触れるユーザーインターフェースはキーボードとディスプレイであり, それに拘るのは当然のことだといえる. しかし, 見方を変えれば慣れの問題ではないだろうか!?　慣れ

ワンラインNEWS
富士通研究所はLANなど大規模な通信ネットワークを作る場合の維持コストを最小にするエキスパートソフトを試作した. 計算が従来に比べ短時間でできるのが特徴.

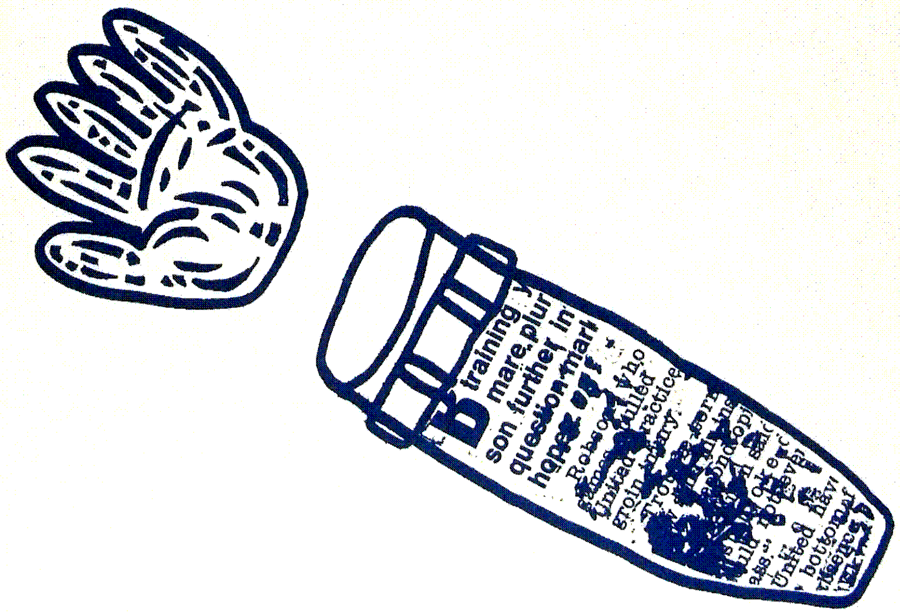
てしまえば，他人はどういおうと当人にとっては一番使いやすいのだ．八方美人のキーボードなど，この世の中どこにも存在しない．

海外では，ユーザーのわがまま（?）に対応できるよう，いろいろなタイプのキーボードが発売されている．「キートップの張り替え可能」「スキャンコードを再設定」「ファンクションキーが多い」等々，多彩な品ぞろえである．国内では，PC-9801用に（株）アスキーがAS Keyboardとして販売している程度で，ほかには見当たらない．もっとおもしろいキーボードが国内でも出てきて欲しいと思う．

3 JIS配置かASCII配置か!?はたまた……でも，バーチャルコードくらい統一したら

最近，パソコン通信でも，「PC-9801 vs DOS/V」の話題がよく上がっている．ここでもやはり，キーボードの問題がいろいろな角度で議論されている．私自身も，PC-9801→J-3100→AX→PS/55→DOS/Vと渡り歩いているため，双方の意見はそれなりに理解できる．別にPC-9801とDOS/Vに限った話ではないが，キーボードに関しては結論が出ないのが現状ではないだろうか!?ASCII配置やJIS配置の違い，さらにプラス・アルファの各差分など，相違点はいくらでもでてくる．また，漢字入力ソフトの起動方法に至っては，[Ctrl]＋[XFER]，[XFER]，[NFER]，[Shift]＋[漢字]，[Alt]＋[半角]/[全角]，[Shift]＋[半角]/[全角]，[漢字]，[Ctrl]＋[Space]，[Space]，[変換]，[無変換]，[英数]/[カナ]……書き出すときりがない．これに，パーソナルワープロまで加えると，わけがわからなくなってしまう．統一しようにも，時すでに遅しといった感じである．

困ったことに，これらの弊害（?）が日本語Windowsにまで及んでいる．最近出てきたWindowsプロセス型の日本語FEPがそうである．ATOK7W，VJE-γなど，本来どのマシンのWindows上でも動くハズなのだが，漢字キーに相当するバーチャルキーのコードが各社まちまちなため，組み込むことはできても起動しないのだ．4月号でも少し触れたが，「Windowsが各マシンの違いを吸収する」という夢は早くも崩れさってしまった．物理的なキー配置は統一できなくても，せめて漢字キーに相当するバーチャルコードくらいは統一できそうなものなのだが……．



Computer Laws

コンピュータの法律

プログラムの改造・その4

熊谷信太郎(弁護士)

■改造されたプログラムの著作権

プログラム購入者と著作権者との間で改造禁止の特約が有効に成立している場合，購入者が勝手にプログラムを改造すると，著作権法違反の問題は生じませんが，債務不履行という契約上の責任は発生することは前回に説明しました．

今回は，その場合の改造されたプログラムの著作権は誰に帰属するかを検討します．

プログラムの改造の問題は，改造の程度によって，法律関係が変わってきます．

改造といっても，小規模のものにすぎず，多少の修正増減を加えただけであり，従来のプログラムと実質的に同一であって，そこに創造性が認められない場合には，改造といっても新たに著作物がつくられたとは評価できませんから，プログラム著作権は従来の著作権者に帰属し，改造者にはなんの権利も発生しません．

■2次的著作物とは

これに対し，改造されたプログラムに創造性が認められる程度の大きな改造がなされた場合には，改造後のプログラムはオリジナルのプログラムの2次的著作物となります．

2次的著作物は，改作著作物ともいわれるものであり，オリジナルの著作物を「翻訳し，編曲し，若しくは変形し，又は脚色し，映画化し，その他翻案することにより創造した著作物をいう」とされています*1.

要するに，2次的著作物といえるためには，オリジナルの著作物を下敷きにしていることと，新たに創造を加えて，オリジナルの著作物とは別の著作物と評価されることという2点が必要なのです．たとえば，小説を原作とする映画が作られた場合を考えると，小説がオリジナルの著作物であり，それを下敷きにして映画を作っているわけですが，原作が同じでも脚本が異なれば映画の印象はまったく違ったものになることからも明らかなように，そこには，原作とは別の著作物と評価しうる創作性が認められるのです．従って，その場合の映画は原作の2次的著作物と評価できるのです．判例では縫いぐるみの「たいやきくん」が絵画の「たいやきくん」の「原画に依拠して，これを変形して製造したもの」と認めたものがあります*2.

■オリジナルと2次的著作物の著作権

プログラムの場合も同様に，下敷きになる原作があって，それに創作を加えて別の著作物を作ったという場合に，改造後のプログラムが2次的著作物と評価されることになります．

そして，改造後のプログラムが2次的著作物と評価された場合には，オリジナルのプログラムの著作権者の権利と，2次的著作物の著作権者の権利の双方が2次的著作物に及ぶことになります．従って，2次的著作物である改造後のプログラムを利用する場合には，オリジナルの著作物の著作権者の権利と，2次的著作物の著作権者の双

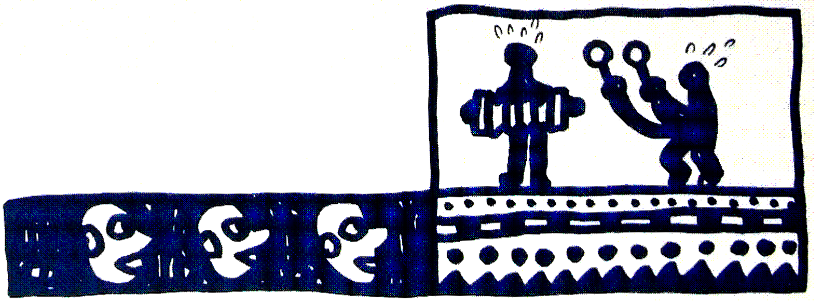


Illustration : Nobuki Fukuhara

ワンラインNEWS
富士通はファクシミリを制御するのに必要な機能をすべてワンチップに組み込んだLSIを開発.これを使うことによりファクシミリに必要な半導体は従来の1/10以下になる.

方の許諾を得る必要がありますが[*3]，もし，許諾を得なかったとしても，作成された著作物は2次的著作物として保護されます．

これに対し，プログラムの改造が極めて大規模で，オリジナルのプログラムと似ても似つかぬものであり，実質的類似性（Substantial similarity）がないという場合には，2次的著作物ではなく，まったく新しい著作物を作成したということになりますから，改造後のプログラムは改造者だけが著作権をもつことになります．

＊1　著作権法第2条第1項第11号
＊2　東京地裁昭和52年3月30日判決
＊3　著作権法第11条

アキバ短信

営業時間　10：00～19：30　年中無休
TEL 03-3255-3451

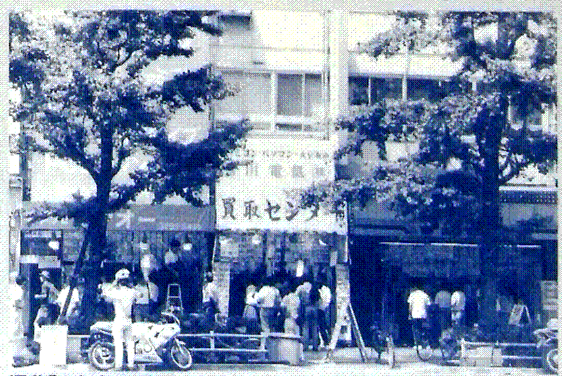
メッセ サンオーII
ファミコンソフトに時代を見た

JR秋葉原駅の改札を出て，中央通りを上野方面に向かってしばらく歩くと，最近テレビコマーシャルにも登場するようになった「ソフマップ」をはじめ，中古のハードウェア，ソフトウェアを取り扱う店が目につく．なかでもT-ZONEの向かい側あたりは，店頭一杯にズラリとファミコンソフトを並べた店が軒を連ねている．任天堂のファミリーコンピューターをはじめ，各種ゲーム機のソフト，ハードを扱っているのだが，そこは秋葉原，新品ばかりとは限らない．どの店にも「中古ソフト買い取ります」の看板が掲げられ，店内の一角には，もはやパッケージも説明書もない裸のファミコンソフトがセロハン紙にくるまれて1000円～2000円くらいで売られている．その店構えがどこも似たりよったりであるせいか，ここは通称「3軒並び」と呼ばれ，かなりのゲームマニアでもたしかな店の名前など知ってはいない．「どの店で買ったの？」と誰かにたずねられたときは，「3軒並びの真ん中」と答えればいいのだから．その流儀でいえば，メッセサンオーは「3軒並びの左端の店」，IIがつくのはその2階にあるからということになる．

メッセサンオーIIは，輸入ゲームソフトやアーケードゲームの基盤を販売している．「輸入ゲームソフト」なるものについてはいささかの説明が必要だろう．グラフィックスの美しさでは抜群のAmigaのソフトなどもあるにはあるが，中心はNESやジェネシスのソフトである．NESはNintendo ENTERTAINMENT SYSTEM，すなわち任天堂のファミコンの輸出バージョン，ジェネシスはセガのメガドライブの輸出品である．国内で販売されているファミコンでNES用ソフトを使用するには，簡単なアダプタを取り付けるだけでよい．

ところが，これらのソフトにまじってとんでもないものが展示されているのを見つけてしまった．「190 IN 1」「115 IN 1」「76 IN 1」，これらはその名前が示す通り，1つのカセットに190本，115本，76本のゲームが収められた海賊版ソフトである．中古品で説明書などはいっさいなし．それでもゲームの本数に0を2つつけたくらいの値段がついている．ご丁寧にも，この店ではそれぞれのカセットにどんなゲームが入っているかというリストも掲示されていた．案の定，最新のゲームは含まれず，中には「バグあり」という印のついた謎のゲームもある．

輸入ソフトにまじっているところを見ると，著作権という意識の薄い香港か東南アジアあたりが出所と思われるが，それにしても200本近いゲームを1つのカセットに納めようという意欲には驚かされる．だが，見方を変えれば，これらの違法カセットは，ファミコンが登場してから現在までの間に，いかにメモリの集積度が向上し，価格が下がったかを証明するものでもある．おととし大ヒットしたファミコンソフト「ドラゴンクエストIV」は約4Mbit程度らしいが，同じ容量のところへファミコン草創期のゲームを詰め込めば，ゆうに100本や200本は入ってしまうということなのだろう．かくしてファミコンソフトも時代を映すのであった．



通称「3軒ならび」

●プログラマの仕事場 7

Programmer's Working Space

株式会社システムハウスミルキーウェイ

公認会計士とパソコンの出会いがビジネスソフトのベストセラーを生んだ

システムハウスミルキーウェイは，「日経パソコン」誌の読者が選ぶベストソフト・業務ソフト部門で過去6回にわたって第1位，金賞を授賞したパソコン用財務会計ソフト「大番頭」の開発・発売元である．オフコンの守備範囲と思われていた事務処理の分野で，パソコン用パッケージソフトである「大番頭」の成功が与えた衝撃は大きい．同社を訪ねて，吉井秀元社長からお話をうかがった．

業務のプロがソフトを作ったらまったく新しい形のものができた

システムハウスミルキーウェイが設立されたのは，1980年1月のこと．「小さなパーソナルコンピュータでも，事務用専用機並みの事務処理ができるのではないか」と考えた，3人のパソコンマニアがはじめた会社だった．その3人のうち，2人までが公認会計士であったことが，同社のその後の製品の性格を決定づけたといえるだろう．同社が設立された1980年前後は，ちょうど日本におけるパソコンの草創期ともいうべき時代で，79年末にはパソコンブームを生み出す原動力ともなった日本電気のPC-8001が発売されている．ミルキーウェイが最初に手がけたのも，PC-8001に対応した財務会計パッケージであった．

「ふだん見ているのは，ゲームのソフトばかりでしたから，そういう使いやすい感覚で操作できる業務ソフトができるのではないか，またできて当然だと思って作ったところ，オフコンや汎用機のシステムとはまったく違ったソフトができたのです．よく『ユーザーオリエンテッド』といわれますが，実務で使う立場から見て，こんな処理が必要ではないかという機能を取り入れたわけです．いまでは，他社さんに基本型をまねされましたので，どのソフトをとっても似たようなものになっていますが」（吉井社長）．

伝票を入力すれば，月次処理を待たなくてもリアルタイムで帳票が見られる．いまではそれが当たり前になってしまったが，最初にそれを提案した当時は，画期的なことであったという．

「財務会計パッケージ」という名称であったこの会計ソフトは，82年にPC-9801が発売になると，翌年8月には「日本語財務会計　大番頭」としてこれに対応．84年には日本経済新聞社年間優秀製品賞を授賞することとなる．

意外に少ないプログラマ全社員中約4割

現在は，日本語財務会計ソフト「三代目大番頭」で有名なミルキーウェイだが，仕入・在庫管理，販売管理，給与計算など各種の業務処理用パッケージソフトも開発，販売している．同社では，受託や請負での開発はいっさい

ワンラインNEWS
CSKは年内にも台湾・中国に進出する．グループ企業を通じて現地企業や各省と合弁会社を設立し，ソフトウェアやシステムウェアの開発を手がける．

行わない．売上のほとんどをこれらのパッケージソフト関連が占めているのである．製品はほぼ100パーセント自社開発．販売中のパッケージソフトのうち，「パートナー」だけは，ある会社がオフコンのソフトをパソコン用に移植したものだが，これも昨年全面的に社内で書きなおしているという．

現在社員数は約100名，そのうち40名弱が開発要員である．4割弱という比率は，自社開発のソフトウェアハウスとしてはかなり低いように思えるが，吉井社長はこの点について次のように説明してくれた．

「営業，サポート，開発を3本柱として位置づけているところが，ほかのソフトウェアハウスさんとは大きく違う点でしょう．業務系のソフトというのは，一般のパソコンショップに置いておくだけでは売れません．ショップと協力しあって，セミナーや講習会を開いたり，こちらから出向いてご説明したりしないと，なかなか買っていただけないのです．ですから，全国を営業マンが回っておりますし，大阪，名古屋，福岡，広島，札幌に営業所を置いています．また，業務系ソフトの場合はワープロなどと違って，使い方がわからないと即業務に支障が出てしまいますから，どうしてもしっかりしたサポートが必要です．そこでサポートにも約3分の1ほどの人員を割いています．」

本社ビル4階にある開発部門の仕事場を見せていただいたが，まず驚くのはそのスペースのぜいたくなことである．各人にはL字型に配置された机が割り当てられ，その広さは机にマシンを2台ずつ置いてもまだ十分に余裕があるほどだ．机は2人ひと組でコの字型の配置になっており，低いパーティションで区切られている．プログラマ同士のコミュニケーションに配慮した結果でもあるという．

プログラマは，4～5人でチームを組んで開発にあたっている．かつてBASICとマシン語でプログラミングしていたころには，作業の切り分けが難しかったが，いまではほぼすべてのプロダクツをC言語に書き換えたため，グループでの開発は楽になったそうだ．机の上に置かれたマシンはその多くが32ビットパソコンでLANによって結ばれている．

業務処理ソフトの開発には実務の知識と高い技術力が要求される

業務ソフトの開発で，最も大きなポイントとなるのはソフトウェアの仕様である．とくに，消費税の導入以降は法律に関するさまざまな処理が入ってきているという．たとえば，減価償却ひとつをとっても，表面的に減価償却の処理を行うその裏には，各種の法律がある．汎用パッケージソフトとしては，細かい法律に対応した処理を取り入れたいが，実際に使用するユーザーが，法律はおろか用語する知らないこともあり得る．「どういったユーザーイ

About Company

吉井秀元社長．1980年1月に設立．現在は財務会計ソフト「三代目大番頭」をはじめ，ビジネス分野での各種パソコン用パッケージソフトの開発・販売およびコンサルティングを行っている．自社のパッケージ製品だけで成り立っている，数少ないソフトハウスのひとつだ．

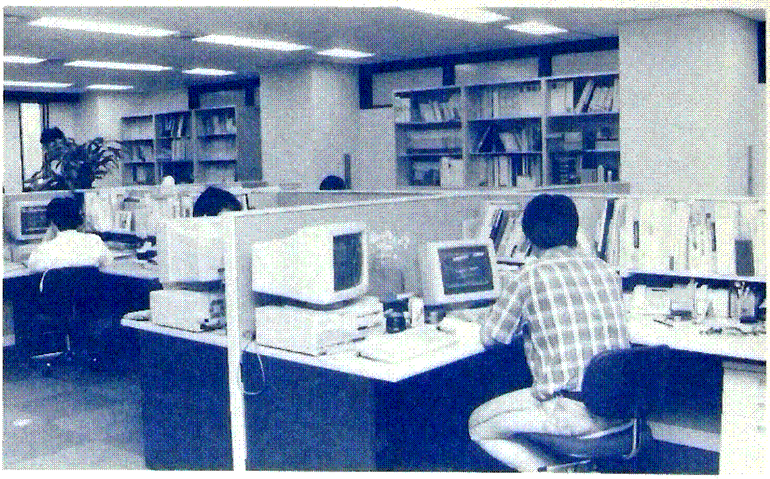
吉井社長

Location

本社は港区赤坂のTBSテレビにほど近い一角で，ビルの1階から6階までを占める．1階はショールーム．札幌，名古屋，大阪，広島，福岡の各地に営業所がある．

Products

「三代目大番頭」(250,000円)
消費税対応型の財務会計システム．伝票を入力した時点で即座に対応する帳簿が画面上で参照できるというパソコンならではの柔軟なユーザーインターフェースを最初に取り入れたことで知られる「大番頭」の最新バージョン．


▲開発部門

「三代目」は，消費税税務を日常会計レベルで取り込んでおり，内税・外税の切り替えや，税込み，税抜きによるアウトプットの切り替えも簡単にできる．人間がデータを追っていく過程をそのままトレースして画面に表示する，ハイパーテキストライクな特徴を持つ．
NEC PC-9800シリーズ，東芝J-3100シリーズ，各社AX，富士通FM-Rシリーズ，松下Panacom Mシリーズ，IBM PS/55シリーズに対応．

「大問屋」（文節変換版78,000円，熟語変換版68,000円）
PC-9800シリーズ用仕入・在庫管理ソフト．仕入・出庫・在庫を一元管理し，同時に買掛金を管理することで，仕入，販売，支払に関わるムダを排除した．大問屋の仕入・出金データは三代目大番頭の仕訳データとして使用することも可能．VJE-Σ，ATOK，松茸対応の文節変換版と，NEC熟語に対応した熟語変換版がある．

「大福帳」（文節変換版78,000円，熟語変換版68,000円）
PC-9800シリーズ用販売管理ソフト．環境設定終了後，ただちに伝票発行処理ができ，新規の得意先，商品，担当者の台帳登録なども伝票発行中に行える．また，日常の伝票入力，請求書発行などの業務を行うだけで，自然に台帳ごとに販売データが蓄積され，いつでも管理資料として出力できる．「三代目大番頭」「大問屋」との間で相互にデータを利用することも可能である．

「パートナー」（180,000円）
PC-9800シリーズ用給与計算ソフト．ハードディスク版では200人，フロッピーディスク版では100人までの給与処理が可能．いったん給与計算の基礎となる賃金規定を登録すれば，専門知識がなくても操作できる．銀行への振込データのフロッピー渡し，「大番頭」へのデータ転送，表計算ソフトからの勤怠データの取り込みなどにも対応．保守契約（初年度はユーザー登録時に自動的に契約，次年度以後は30,000円／年で更新可能）を結べば，税制改正にも迅速に対応する．

ンターフェースをとりながら組み込んでいくのかが今後の検討課題ですね」と吉井社長は語る．

実際のソフト開発にあたっては，基本設計，詳細設計，プログラミングという，基本的な手順はあるが，頭の中だけで設計したインターフェースは，いざ実際に使うとなると具合の悪い点も出てくる．そこで，プロトタイプを作成し，それを検討しながら繰り返し改良を重ねているという．

開発するソフトウェアの性格から，プログラマは会計業務についての知識も求められる．もちろん，ソフトウェアの設計や実際のプログラミングにはプログラマとしての技術的な知識は必須である．加えて，業務ソフトとしての使い勝手の善し悪しを評価し，どのように改良したらよいかを提案するシステムエンジニア的なセンスも必要だ．「学校で学んできた知識だけでは，うちのようなソフトウェアは作れません．業務処理だからといって，COBOLのプログラムを作るようなわけにはいきません．むしろ，表計算ソフトやワープロを作るのと近いようなプログラムを書くわけですから．もちろん，業務処理や法律も覚えなくてはなりません．それに，『こんな風にすれば，わからない人でも使えるようになるんじゃないか』といった提案が必要ですから，かなり総合的な知識が要求されますね．何事にも興味をもって対応できる人にきてほしいです．」（吉井社長）

パソコンのビジネスソフトは個人の情報武装ツール

業務ソフトの開発というと，かなり地味なものを想像しがちだが，LANへの対応，使いやすいユーザーインターフェースの研究など，パソコン分野の新しい技術を取り入れる積極的な姿勢はほかの分野のソフト開発と変わりはない．

最近では，「コンピュータの情報との接し方」としてMacintoshのHyperCardに注目して，「大番頭」にハイパーテキスト的なインターフェースを取り入れようとしている．HyperCardではオーサリングツールを使って，紙芝居のように次々と画面表示させることができる．業務ソフトのユーザーが，「大番頭」を使って作成した管理資料を検証するにあたって，HyperCardのように次々と資料を画面に表示して検索できるような機能が必要になってくるだろうというのである．

伝票入力中に財務諸表を参照したり，呼び出したい元帳の行に合わせれば，すぐ個別の元帳を参照できるようにしたり，元帳のデータを月別，日別で見られるようにしたりといった機能はすでに取り入れているが，もっと大きい管理資料を作る場合にも，こういった細やかなインターフェースを取り入れることを考えているという．このような考え方を経理だけでなく，販売管理システムにも適用していくことで，オフコンや汎用機のアプローチとは一線を画し，パソコンを個人の情報武装ツールとして提案していきたいというのが，吉井社長のビジョンである．「オフコンでは，情報として使えるデータがたくさん寝ているのです．個人個人が情報を使いこなせる情報武装した中小企業が，これからの日本を引っ張っていくのではないでしょうか．」

ワンラインNEWS
東京電機大学は英単語の意味をコンピュータで検索したり，正しい発音をスピーカーから確認できる音声付き英語辞書「サウンディクショナリー」を開発した．

●イベント

■9月25日(水)~9月27日(金)
第6回IC・光カード&ASCAT展
ICカード、光カードの最新技術と関連機器の展示。
セミナー、「SCAN-TECH JAPAN '91」も同時開催される。
会場：東京流通センター・第2展示場(東京・平和島) 入場料：1,000円
問い合わせ先：日本工業新聞社事業部 TEL 03-3292-3561

■10月1日(火)~10月5日(土)
1991年エレクトロニクスショー
国内外のエレクトロニクス関連製品、最新技術を展示。
10月1日~3日までは招待日、4日~5日は一般公開日。
会場：日本コンベンションセンター(幕張メッセ)
入場料：500円(一般)、300円(学生)
問い合わせ先：日本エレクトロニクスショー協会 TEL 03-3284-1051

■10月2日(水)~10月4日(金)
ソフトウェア・ショウ'91
ホビーソフトを除く、大型コンピュータからパソコンまでのソフトウェアを展示。
10月2日は招待日、3日~4日は一般公開日。
会場：池袋サンシャインシティ・センターホール(東京) 入場料：無料
問い合わせ先：ソフトウェア・ショウ事務局 TEL 03-3437-3394

■10月9日(水)~10月11日(金)
マルチメディア国際会議'91
マルチメディアに関する講演と関連製品の展示。
会場：日本コンベンションセンター(幕張メッセ)
受講料：85,000円(3日間共通、入場料・昼食代含)
問い合わせ先：マルチメディア'91運営事務局 TEL 03-3542-1404

■10月10日(木)
第40回オーディオフェア
コンピュータとシンセサイザーによるデジタルサウンド・コンテストが行われる。
作品の募集締め切りは10月3日。
会場：池袋サンシャインシティ(東京)
問い合わせ先：(学)尚美学園「デジタルサウンド・コンテスト」事務局 TEL 03-3818-7691

■10月22日(火)~10月25日(金)
データショウ'91
"人にやさしい情報環境の創造"をテーマに
164社がハードウェア、ソフトウェアを展示。
会場：東京国際見本市会場(東京・晴海) 入場料：無料
問い合わせ先：データショウ事務局 TEL 03-3433-4547

●海外イベント

■10月21日(月)~10月25日(金)
COMDEX/Fall '91
最大規模のコンピュータ関連製品の展示会。約1800社が出展。
会場：Sands Expo and Convention Center(米国・ネバダ州ラスベガス)
問い合わせ先：ザ・インターフェイス・グループ日本支社 TEL 03-3237-875

information・page

●セミナー

■10月8日(水)~10月9日(水)
オブジェクト・オリエンテッドテクノロジー入門セミナー
オブジェクト指向による分析設計プログラミングからその長所短所までを解説する。
会場：麹町会館(東京) 受講料：96,000円(3名以上割引あり)
問い合わせ先：ソフトウェアテクノロジーインターナショナル TEL 03-5689-3536

■10月11日(金)、10月25日(金)、11月8日(金)
先端技術基礎講座
「プロになるための計測・制御・画像処理技術」
計測・制御・画像処理技術に関する講習会。申込み締め切りは10月4日。
全3回、各回のみの参加も可。
会場：龍生堂ホール(東京・新宿)
受講料：70,000円(会員、全3回)、25,000円(会員、1回のみ)
110,000円(非会員、全3回)、38,000円(非会員、1回のみ)
問い合わせ先：(社)精密工学会 TEL 03-3362-1979

■10月31日(木)
386ネイティブCコンパイラ実践プログラミングセミナー
386/486をフルに使いきるプロテクトモードプログラミングなどをWATCOM C/386を例に解説。
会場：東京YMCAホテル(東京・千代田区)
受講料：20,600円(消費税・資料・昼食代含)
問い合わせ先：(株)ライフボート「386ネイティブCコンパイラ実践プログラミングセミナー」係
TEL 03-3293-4714

Microsoft University

■9月30日(月)~10月4日(金)、10月14日(月)~10月18日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
会場：三田会場 受講料：180,000円(5日間)

■10月14日(月)~10月18日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための入門コース。
会場：三田会場 受講料：180,000円(5日間)

■10月21日(月)~10月25日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
会場：三田会場 受講料：180,000円(5日間)

■10月21日(月)~10月25日(金)
MS OS/2 Applications Programming
OS/2アプリケーション開発のための上級コース。
会場：三田会場 受講料：180,000円(5日間)

■10月2日(水)~10月4日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
会場：三田会場 受講料：90,000円(3日間)

■10月7日(月)~10月8日(火)
C言語入門
Quick Cを使ったC言語の入門コース。
会場：三田会場 受講料：60,000円(2日間)

■10月15日(火)~10月18日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース。
会場：三田会場 受講料：150,000円(4日間)

■10月21日(月)~10月25日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース。
会場：三田会場 受講料：180,000円(5日間)

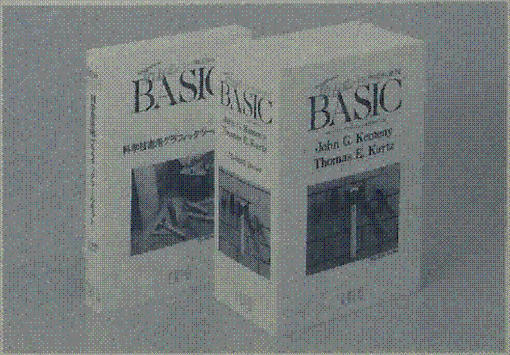
Microsoft Universityは、OS/2、Windowsアプリケーション開発者のために定期的に開催されているセミナーです。なお、会場が変更となる場合がありますので、下記へご確認ください。
問い合わせ先：富士ソフトウェア(株) MU教育センター TEL 03-3452-7722

NEWS & NEW PRODUCTS

クレオ，構造化プログラミングが可能な「TrueBASIC」を発売

（株）クレオ（パソコン事業部：TEL 03-3456-4615）は，従来のBASICの機能を拡張・強化し，構造化プログラミングの理論を融合させた「日本語TrueBASIC V3.0」を発売した．ANSI規格に準拠し，ループを脱出するFOR EXIT文や条件判断時のSELECT CASE文などにより，構造的な制御構造を形成することができる．サブルーチンは，パラメータとして変数や値を受け取ることができ，変数はローカル変数としても，繰り返し使うこともできる．フルスクリーンエディタ，テキスト用・グラフィックス用のマルチウィンドウをサポートし，コンパイルにより，コンパクトで高速な中間コードを形成することができるなどの特徴をもつ．

ビジネス用グラフィックス，数学者用，3次元グラフィックス，科学技術用グラフィックス，統計用グラフィックスなどのツールキット（15,000円～25,000円）も発売の予定．対応機種は，NECのPC-9800シリーズと98NOTE，価格は18,000円．



ライフボート，本格的COBOLインタープリタを発売

（株）ライフボート（TEL 03-3293-4711）は，カナダWATCOM社が開発した本格的なCOBOLインタープリタ「WATCOM COBOL」を発売した．機種依存部分を取り去り，同社独自のインターフェースとエディタを付加している．COBOLの標準規格に準拠し，COBOLの機能単位のうち，中核，表操作，順ファイル，索引ファイルのレベル１を完全にサポート，レベル２の機能の主要部分もサポートしている．第２種情報処理試験の出題範囲をほとんどをカバーしているため，COBOLプログラミングの実習にも適している．対応機種は，PC-9800シリーズ（ノーマルモードのみ），J-3100シリーズ，FMR50/60/70で，価格は58,000円．また，ライフボートではWATCOM社と契約を結び，同社の言語製品を日本のMS-DOSマシン用に販売していく．

創夢，X Window X11R5関連製品の提供・サービスを開始

(株)創夢（TEL 03-5453-1251）は，ワークステーションの標準的なウィンドウシステム「X Window System Version11 Release 5」関連製品の販売，テクニカルセッションなどの各種サービスを開始する．製品は，コマンドリファレンス，Xlib-C言語Xインタフェースなどの「X11R5テクニカル・ドキュメント・シリーズ」，移植ガイド，リリース・ノートなどの「技術資料シリーズX11R5関連」，ユーザ・コマンド，Xlib関数などをmanコマンドで参照可能な「X11R5日本語オンライン・マニュアル」など．

ラインズコーポレーション，教育用「Mind WAVE」を発売

（株）ラインズコーポレーション（公共システム部：TEL 0422-49-7233）は，日本語プログラミング言語「Mind5.5E」（教育用Mind）とオーサリングツール（教材作成支援ソフト）「MIND WAVE」をパッケージにした教育用ソフトウェア「Mind WAVE」を発売した．

「Mind5.5E」は，学習がしやすく，学校における言語教育やプログラミングに適しており，

ワンラインNEWS
日本電気はスーパーコンピュータ並に高速演算のできるワークステーション「スーパーワークステーション」の開発に着手した．数年後にパソコンと並ぶ主力製品に育てる．

NEWS & NEW PRODUCTS

「MIND WAVE」は，Mind言語をほとんど知らなくても，文章・図形・アニメの各画面モードで作られた部品を組み合わせ，あらかじめ用意されている機能語を用いるだけで簡単に教材を作成することができ，先生の教材開発を強力に支援する．PC-9800シリーズとFMRシリーズとの完全互換，アイコン表示，マウス入力などのユーザーインターフェースを備えている．価格は，基本セットが50,000円，サイトライセンス契約による学校セットが450,000円．

FREIA，LASERSHOT用C言語インターフェースライブラリを発売

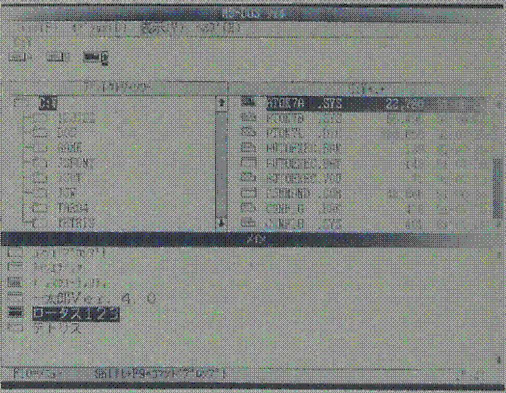
（株）FREIA（TEL 0474-70-1064）は，キヤノン製レーザプリンタ「LASERSHOT」（LBP-A406/B406/B406S/B406D/A404）で採用されているページ記述言語LIPSを，C言語から利用するためのインターフェースライブラリ「CLIPS」を発売した．LIPSII，II＋，IIIの機能を関数レベルでフルサポートし，今まで困難であったC言語からのLASERSHOTの制御を容易にすることができる．従来のLIPSの機能ではいくつもの工程を必要としたベクタモードにおける文字出力なども，それぞれの機能についての関数のほか，拡張関数で簡単に出力することができる．DOS版とUNIX版があり，それぞれソースコードを添付している．DOS版の個人購入価格は38,000円，法人購入価格は114,000円，パッケージ供給ライセンスは300,000円（数量制限なし）．UNIX版の価格は174,000円（個人ユーザーには1/3の価格で提供），パッケージ供給ライセンスは300,000円（数量制限なし）．また同社では，CLIPS応用アプリケーションも発売する予定．

三洋電機，MS-DOS 5.0を国内初製品化，バンドルマシンを発売

三洋電機（株）（情報システム事業本部営業統括部コンピュータ企画部：TEL 06-443-5145）は，国内ではじめてMS-DOS 5.0を日本語化し，これをバンドルした32ビットノートパソコン「AXÂGE NOTE386SX（MBC-18NVH2/NVH4）」を発表した．MS-DOS 5.0は，ユーザーズメモリエリアの拡張，グラフィカルな画面表示のDOSシェル，大容量（最大2GB）ハードディスクのサポート，アプリケーションをワンタッチで切り替えることができるタスクスイッチ機能などを備えている．また，このマシンは，AX協議会が定めたビデオシステムAX-VGA/Sを採用している．

■おもな仕様

プロセッサ：80386SX（20MHz）／メモリ：3MB（最大5MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・20MB HDD×1（NVH2），3.5インチFDD×1・40MB HDD×1（NVH4）／ディスプレイ：CFLバックライト付白黒液晶／価格：368,000円（NVH2），398,000円（NVH4）



富士通，486SX搭載の高解像度デスクトップマシンを発売

富士通（株）（FMインフォメーションサービス：TEL 03-3646-0816）は，高解像度機としては国内ではじめて80486SXを搭載した，デスクトップパソコン「FMR-80HE1」を発売した．大容量ハードディスクを搭載し，LANシステムでのサーバやOS/2でのホスト連携システムなどに適している．また同

NEWS & NEW PRODUCTS

時に，従来機の「FMR-50HV/FV」のグラフィック機能，拡張性を向上させた中解像度マシン「FMR-50HL1/FL1」（410,000円～660,000円）も発売した．

■FMR-80HE1D7/D1/C7/C1

プロセッサ：80486SX（20MHz）／メモリ4MB（最大16MBまで拡張可能）／外部記憶装置：5インチFDD×2・170MB HDD×1（D7）・100MB HDD×1（D1），3.5インチFDD×2・170MB HDD×1（C7）・100MB HDD×1（C1）／価格：1,420,000円（D7/C7），1,220,000円（D1/C1）

日本サン・マイクロシステムズ，デスクトップSPARCstationを強化

日本サン・マイクロシステムズ（株）（TEL 03-3221-7228）は，SPARCstationのコストパフォーマンスを強化した新機種「SPARCstation ELC」，「同IPX」を発売した．ELCは，RISC/UNIXワークステーションとしてはじめて80万円台を実現し，従来のSLCにくらべて約2倍のCPU性能をもち，最大4倍のメモリを搭載することができる．IPXは，40MHzの新設計SPARCプロセッサを搭載し，24.2SPECmarks，28.5MIPS，4.2Mflopsを実現している．従来2つのSBusスロットを占有していたGXグラフィックス・アクセラレータを1チップ化してシステムボード上に標準搭載している．また同時に，ディスク，メモリを2倍に拡張し，1280×1024ピクセルの解像度をサポートする「同2GXplus」を発売，「同IPC」を約20%の値下げを行った．

■おもな仕様

プロセッサ：SPARC（33MHz，ELC）同（40MHz，IPX）／メモリ：8MB（ELC，最大64MB），16MB（IPX，最大64MB）／外部記憶装置：3.5インチFDD×1・207MB HDD×1（IPX）／価格：891,000円（ELC），2,695,000円～（IPX）

エプソン，最大7台のSCSI機器接続可能なSCSIボードなどを発売

セイコーエプソン（株），エプソン販売（株）（エプソンインフォメーションセンター：03-3377-3500）は，SCSIボード「エプソンSC-1000V」を発売した．このボードは，NECの「PC-9801-55L/U」のアッパーコンパチを実現し，ハードディスク，MOディスクなどのSCSI対応機器を，OS，デバイスドライバを組み合わせることにより，最大7台までデイジーチェーン接続することができる．価格は，30,000円．また同時に，「エプソンPC-286Lシリーズ」，「同PC-386BOOK L」などで採用しているLスロットに装着して使用することのできるSCSIボード「エプソンPCLSCB」（30,000円）も発売した．

スピリット，DOS/V用マルチスクリーンエディタを発売

スピリット（株）（TEL 03-3234-8041）は，DOS/Vマシン用マルチスクリーンエディタ「VISP」を発売した．プログラムの編集中に自由にC言語のシンタックスチェックが行え，コンパイルとエディットを繰り返すことがなく，ソフト開発の生産性を向上することができる．同時に10個のファイル編集が可能で，128コマンドレベルのアンドゥ機能，200以上のコマンドなどをもつ．対応機種は，IBM PC/ATと互換機などのDOS/Vマシン．価格は38,000円．

ワンラインNEWS
ティアックは厚さが1/2インチと業界最薄の3.5インチFDD「FD05」シリーズを開発，8月よりサンプル出荷を開始した．従来は三菱のFDDが14.8ミリと最も薄かった．

NEWS & NEW PRODUCTS

アドミラルシステム，98，DOS/V対応のバックアップツールを発売

（株）アドミラルシステム（営業部第１営業課：TEL 0482-52-4646）は，同社のNewtonシリーズの第２弾として，PC-9800（98NOTE対応）とDOS/V用にディスクバックアップツール「Newton-Backup」を発売した．バックアップ／リストアしたいファイルを画面の中から簡単に指定することができ，その際にバックアップ管理メモを付加することができるため，リストア時にどのバックアップをリストアするかを管理メモの一覧を表示して選択することができる．MS-DOSフォーマットを採用しているため，フロッピー以外のディスクにもバックアップが可能で，未フォーマットのフロッピーディスクでも自動的に内蔵フォーマッタがMS-DOS形式にフォーマットする．そのほか，ファイル圧縮機能，RAMドライブやMOディスクなどからのバックアップ，バックアップしたドライブと異なるドライブへのリストアなどが可能．DOS/V版では日本語，英語両モードに自動的に対応している．価格は9,700円．

アスキー，パソコン用UNIX学習ソフトを低価格で発売

（株）アスキー（営業本部：TEL 03-3486-8080）は，パソコン上でワークステーション用OS，UNIXを学習するためのソフト「MINIX Ver.1.5」を発売した．「MINIX」は，UNIXと同じ動作をするように作られているため，パソコン上でUNIXの環境を体験し，簡単にマルチタスク・マルチユーザー環境を学習することができる．ユーザーのレベルに合わせた段階的な学習が可能で，コマンドは175以上，elvis（vi）やelleなどのエディタも付属している．また，ソースプログラムも公開されているため，UNIXの内部構造を解析することができる．対応機種はNEC PC-9801シリーズで，価格は14,800円．

ボーランド・インターナショナル，アシュトンテイト社を吸収合併

米国ボーランド・インターナショナル社と米国アシュトンテイト社は，ボーランド社によるアシュトンテイト社の買収に関する合併契約が合意に達したことを発表した．この契約についてボーランド社の会長，社長兼CEOのフィリップ・カーン氏は「ボーランドとアシュトンテイトは営業戦略に意見の一致を見ており，今後は90年代のカスタマーニーズに応える幅広いテクノロジーとプロダクトをもつすぐれた組織をもつことになるでしょう．この契約によって，我々は，データベース，スプレッドシートからグラフィックス，プログラム言語にいたるまで，あらゆる種類のソフトウェア・プロダクトを提供できることになります．すべてのプロダクトは，あらゆるプラットフォーム上のクライアントサーバコンピューティングのアーキテクチャのために設計されています．我々は，ParadoxやdBASEなど，両社の製品に対するカスタマーの投資を無駄にすることなく，未来のテクノロジーへとスムーズな移行をしていっていただきたいと考えています．」と語った．なお，合併完了後は，現アシュトンテイト社の代表取締役員は全員退任し，ボーランド社の代表取締役会が新合併会社の運営にあたる．

なお，今回の吸収合併について，８月７日に日本で両社の合同記者会見が行われた．席上では，ParadoxとdBASEをはじめとする両社製品に対するサポートを今後も行っていくことが強調された．また，日本において（株）ボーランドジャパンとアシュトンテイト（株）の組織形態をどうするかは，まだ白紙の状態であるという．今後，米国政府による合併承認を経て，合併契約が完了してから，新会社のインターナショナル部門の見直しが行われるのではないかと思われる．

Programmer's Program

MS-DOSユーザーのための UNIX学習ツール

Mortice Kern Systemsツールキット

140をこえるUNIX互換コマンドを実現

ワークステーションは，ここ数年でハイエンドパソコンと重複する価格帯にまで下がってきた．データを共有化するために，ワークステーションをパソコンの代わりに導入する企業も増えている．もう，仕事とUNIXは無縁とはいえないのである．

UNIX関連の書籍だけで学習を試みても，実際にUNIXに触れながら学ぶのに比べると，理解までの道のりは険しいといえる．しかし，いくらワークステーションの価格が下がったとはいえ，まだ個人では手の届かない値段である．

「Mortice Kern Systemsツールキット」(以下MKSツールキット)は，140を越えるUNIX SYSTEM V互換のコマンドをMS-DOS上で実現した，UNIX環境構築ツールである．そのMKSツールキットが日本語に対応し，PC-9800版としてパーソナルメディア(株)からリリースされた．

MKSツールキットは，IEEEがUNIX類似OSの標準化のために提唱している規格であるPOSIXに準拠しており，米国ではAT&Tやヒューレットパッカード社など，多くの企業がUNIX学習用として導入している．使い慣れたPC-9800シリーズ上で，UNIXとほぼ同じ環境が実現できるので，MS-DOSユーザーのUNIX学習用としては最適のツールといえる．

COMMAND.COMを置き換えるコーンシェル

MKSツールキットを使い始めるにあたって，インストレーション(導入作業)とコンフィギュレーション(環境設定)の2つを行う必要がある．

インストレーションは，マスターディスクの1枚目(「Mortice Kern Systemsツールキット 1 of 4」と書かれたディスク)を挿入し，MS-DOSのコマンドラインから

```
A> INSTALL
```

と入力する．あとは対話形式でインストールするディレクトリを指定したり，指示に従ってマスターディスクを入れ換えていけば作業は完了する．

次はコンフィギュレーションである．これも同様にマスターディスクの1枚目を挿入し，MS-DOSのコマンドラインから

```
A> CONFIG
```

と入力すれば，対話型形式で作業が行える．

コンフィギュレーションによって「Mortice Kern Systems KornShell」をCOMMAND.COMの代わりにインストールすることにより，UNIX環境とほぼ同等の機能を実現する設定となる．

マニュアルは，Reference，ユーザーズガイド，インストールガイドの3冊から構成されている．この中でReferenceだけは英文である．ユーザーズガイドは約300ページで，UNIXの環境に触れたことのない人を対象に，各コマンドを学習しながら進められるよう構成されている．インストールガイドは約60ページで，それぞれの作業の内容やQ&Aが詳しく解説されている．

MS-DOSの標準シェルであるCOMMAND.COMの機能は，MS-DOSで環境の進化を妨げる一因になっている．テンプレートも直前に入力したコマンドラインを再現できるだけだし，バッチファイルは単純な分岐やジャンプができるだけで，「スクリプト実行」と呼ぶにはほど遠い．

MKSツールキットは，COMMAND.COMに代わるシェルとして，UNIXのコーンシェル互換の「Mortice Kern Systems KornShell」を用意している．このシェルは，複雑な条件分岐や条件反復が可能で，変数の自由度も高い．また，UNIXのコマンドライン同様に，ワイルドカードの展開もシェルによって行われる．ほかにも，エイリアスやヒストリなど

ワンラインNEWS
ジャストシステムは同社の図形ソフト「花子 Ver.2.0」で利用する部品ライブラリー「中学・高校生物教材資料集」(監修旺文社)を発売した．

の機能が準備されている。

ed, vi, awkなど，豊富なユーティリティ

ls, cp, mv, cat, pgなどの汎用ツールはもちろん，sed, grepなどのフィルタを活用して，パイプやI/Oリダイレクションを組み合わせれば，複雑な処理もUNIX同様にこなすことができる。UNIXの標準ラインエディタであるedや，フルスクリーンエディタであるMortice Kern Systems-vi，そして4GL(4th Generation Language)とも呼ばれるMortice Kern Systems-awkも含まれている。

awkは，C言語ライクなプログラミングのできる汎用テキスト処理ツールである。一般的には，レポート作成，プロトタイプ作成用，情報検索に使われる。awkは，ファイルの内容を読み，選択された情報を獲得し，指定されたフォーマットで情報を提示する。C言語でプログラミングするより簡単に，データ処理プログラムを作成することができるので，用途は数知れないほどだ。

MKSツールキットは，UNIXそのものではなく，MS-DOS上でUNIXと互換性のあるコマンドを提供している。このためシングルタスクであり，ファイル名の制限も取り除くことはできない。しかし，これらを除けば，ほぼUNIXそのものといった環境が得られる。

UNIXの学習を始めようとしているパソコンユーザーだけでなく，UNIXマシンを使っているパワープログラマがパソコン上で同じ環境を実現するためにも，十分役に立つだろう。

■Mortice Kern Systemsツールキット

販売：パーソナルメディア（株）（TEL.03-3490-8841）／価格：68,000円（税別）／動作環境：日本電気PC-98/9800（PC-98XA，LTを除く）・エプソンPC-286/386（IBM-PC用はIBM PC/XT/AT）／必要メモリ：640KB／動作OS：MS-DOS Ver.3.1以上（IBM-PC用はVer.3以上）／供給メディア：5または3.5"2HD（IBM PC用は5または3.5"2DD）／ファイル総量：約3MB

コマンド一覧

alias	diffh	if	pwd	trap
asa	diff3	init	r	true
awk	dirname	integer	read	: (colon)
awkl	dot	id	readonly	tty
banner	du	jobs	return	type
basename	echo	join	rev	typeset
bdiff	ed	kill	rm	ulimit
break	env	lc	rmdir	unalias
c	eval	let	sed	uname
cal	exec	line	set	uncompress
calendar	exit	login	sh	unexpand
case	egrep	logname	shift	uniq
cat	expand	look	she	unpack
cd	export	ls	shv	unset
chdir	expr	m4	size	unstrip
chmod	false	mailx	sleep	until
cmp	fc	mkdir	sort	uudecode
comm	fg	mktemp	spell	uuencode
compress	fgrep	more	split	validfnam
continue	file	mv	strings	vi
cp	find	nl	strip	wc
cpio	fmt	nm	sum	whence
crypt	fold	od	switch	which
ctags	for	pack	sync	while
cut	getopt	passwd	tail	who
date	glob	paste	tar	xargs
dc	grep	pcat	tee	
dd	gres	pg	test	
deroff	hash	posixcon	time	
dev	head	pr	times	
df	help	prof	touch	
diff	history	ps	tr	

オンラインソフト採集帳 10
NECの独自文字をJIS文字に変換する DENEC

PC-9800シリーズや文豪シリーズなど，日本電気製のパソコンやワープロでは，JISで定義されていない数字や罫線記号などが使える．これらの文字を使ったテキストファイルは，日本電気製のパソコンやワープロ間ではまったく問題なく使用できる．しかし，パソコン通信やフロッピーディスクなどを通して，他機種を利用しているユーザーにこのテキストファイルを渡すと，文字が読めないなどの問題が生じる．

これは，JISの情報交換用漢字符号系（JIS X 0208-1983）において，図形文字用符号表の中で9区から15区までが未定義となっており，このうちの9区（29xx）から13区（2Dxx）の範囲に，日本電気が独自に定義した文字コードが含まれているからである．

DENECは，指定ファイルまたは標準入力から，日本電気独自の文字コードをJIS（JIS X 0208-1983）で定義された文字コードに変換して，標準出力に出力するフィルタである．

PC-PR系プリンタで2バイト半角罫線を印字

DENECで，テキストファイルTEST.DOCの内容を変換し画面に表示したいときは，MS-DOSのコマンドラインから，

```
A> DENEC TEST.DOC
```

と入力する．さらにリダイレクトすることにより，ファイルに落すこともできる．

DENECには，次のような起動オプションが用意されている．これらのオプションは，ハイフンではじまるオプションを，ひとつひとつ空白で区切って指定しても，ハイフンのあとにオプション文字を連続して指定してもよい．

- a　罫線記号をアスキー文字の組み合わせに変換
- c　ファイルとしてオープンできなくても処理を継続する
- i　2バイト文字に不正なコードがあっても処理を継続する
- p　2バイト半角罫線記号のみアスキー文字の組み合わせに変換

pオプションは，2バイト半角罫線のフォントをサポートしていないPC-PR系のプリンタでの印字を可能にし，印字する際の全角と半角の比を保つことができる．

PC-9800シリーズではJIS罫線が表示できない

DENECを使って日本電気独自の文字コードを変換したあと，PC-9800シリーズなどで罫線など，読めない文字が出ることがあるので注意が必要である．

これは，PC-9800シリーズがいまだに旧JIS（JIS C 6226-1987）に準拠しているためで，プログラムのバグではない．

DENECでは，現在のJIS罫線に変換する．罫線は8区の罫線素片に変換され，PC-9800シリーズの機種では，このJISで定義されている8区の罫線素片が定義されていないため通常では読めない．ただしエプソンのPC-286/386シリーズや，他のMS-DOSマシン

リスト1　日本電気独自のコード

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥]^_`
abcdefghijklmnopqrstuvwxyz{|}~。「」、・ヲァィゥェォャュョッーアイウエオカキクケコサシスセソタチツ
テトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン゛゜

─│┌┐┘└├┬┤┴┼━┃┏┓┛┗┣┳┫┻╋
①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳ⅠⅡⅢⅣⅤⅥⅦⅧⅨⅩ
㍉㌔㌢㍍㌘㌧㌃㌶㍑㍗㌍㌦㌣㌫㍊㌻㎜㎝㎞㎎㎏㏄㎡ ㍻ 〝〟№㏍
℡㊤㊥㊦㊧㊨㈱㈲㈹㍾㍽㍼㍻≒≡∫∮∑√⊥∠∟⊿∵∩∪∮□
```

リスト2　DENECで変換後のテキスト

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥]^_`
abcdefghijklmnopqrstuvwxyz{|}~。「」、・ヲァィゥェォャュョッーアイウエオカキク
ケコサシスセソタチツ
テトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン゛゜ヰヱヮヵヶヴガギグゲゴ
ザジズゼゾダヂヅデドバパビピブプベペボポ＝ “” …
─││═══════════┌══┌┐══┐└══└┘══┘├├══├══├┤┤══┤
══┤┬══┬┬══┬┴══┴┴══┴┼══┼═
═┼═════════┼‘’“”（）〔〕〈〉《》【】・─││═════════┌
══┌┐═
═┐└══└┘══┘├├══├══├┤┤══┤══┤┬══┬┬═
═┬┴══┴┴══┴┼══┼══┼═════════┼
(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)(17)(18)(19)(20) I II I
II IV V VI VII VIII IX X
ミリキロセンチメートルグラムトンアールヘクタールリットルワットカロリードルセント
パーセントミリバールページmmcmkmmgkgcc平方メートル＝平成 “” No.K.K
TEL(上)(中)(下)(左)(右)(株)(有)(代)明治大正昭和≒≡∫＝Σ√⊥∠＝＝∵∩∪＆□
```


ワンラインNEWS
ソニー・テクトロニクスは米テクトロニクス社が開発した，紙を選ばないインクジェット方式のカラープリンタの販売を開始した．

では問題なく読むことができる.

PC-9800シリーズでJIS罫線など，新JISに対応したコードを見るためのユーティリティが，いくつかフリーソフトウェアとして発表されている．serow氏のjis90，Danna氏のchf，Miko. Hoshina氏のVmult（びゅうまるち）などがそれである.

ソースファイルもあり他機種へ対応が可能

DENEC. LZHの中には，DENECのソースファイルも圧縮されて収められている．プログラムは，エル・エス・アイ ジャパン（株）のLSI C-86試食版（Version 3.2）によってコンパイルされているが，マイクロソフトの日本語対応関数が備わっているCコンパイラなら，他機種への移植も可能である.

なおこのソースファイルは，文字コードに依存した記述がされている．漢字コードにシフトJISを採用していない機種では，インクルードファイルの「OHCTA. H」を書き換える必要がある.

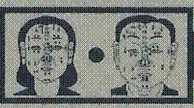
■DENEC Ver.1.00 rev.1.6

名称：DENEC／分類：フィルタ／登録名：DENEC.LZH／作者：八丁堀（ASCII-NET PCS pcs18303）／動作環境：漢字コードにシフトJISを採用しているMS-DOSマシン／登録月日：1991年5月7日／復元方法：LHA E DENEC／入手場所：ASCII-NET PCS（pool msdos #3908）

ファイル構成

```
Listing of archive : DENEC.LZH

  Name          Original    Packed  Ratio   Date      Time    Attr Type  CRC
--------------  --------  -------- ------ -------- -------- ---- ----- ----
  DENEC.C           8216      3365  41.0% 91-04-20 23:06:58 a--o -lh5- E00B
  DENEC.DOC         6601      2746  41.6% 91-04-20 23:13:52 a--w -lh5- 2FAB
  DENEC.EXE        11004      6953  63.2% 91-04-20 23:03:32 a--w -lh5- 4156
  DENEC.FIN         5449      2693  49.4% 91-04-20 23:13:48 a--w -lh5- 5531
  OHCTA.H           3712      1134  30.5% 91-05-04 23:27:50 a--o -lh5- EB73
--------------  --------  -------- ------ -------- --------
     5 files       34982     16891  48.3% 91-05-07 05:25:16
```

HEALTHY?  UNHEALTHY?

⑩ Cばかりがなぜもてる?

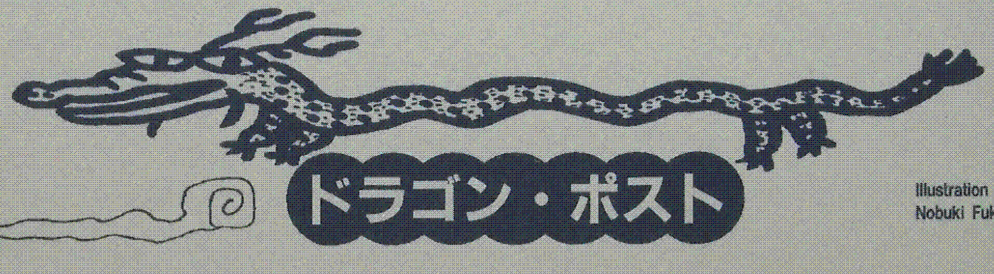
Cといってもじ言語ではない．ビタミンCの話.

機能性飲料のブームは，やや落ちつきを見せてきたが，一般の清涼飲料水にもセンイやフラクトオリゴ糖，ビフィズス菌などが入った製品が多くなってきた．なかでも，機能性飲料ブームの前から相変わらず人気のあるのがビタミンCだ．「レモン○個分のビタミンCが……」というように，レモン＝ビタミンCというイメージを利用した製品が多く，製品のコンセプト自体を支える存在にまでなっている．ビタミンCは，レモン→さわやか→健康というイメージを具現化するものなのだ．毛細管など体内の結合組織の代謝を活発にするという，C本来の動きは二の次になってしまった.

そもそも，レモンはビタミンCを含んだ食品としてそれほどすぐれたものなのだろうか．皮まで含んだレモンには，90mg（100g中，以下同）のビタミンCが含まれている．果汁には45mg．1個あたり50gの果汁がとれるとしても，1日のビタミンC必要量である150mgをとるには，6個分以上の果汁が必要になる．現実にレモンを使うのは，料理に絞りかける程度，ビタミンC源として期待できるものではない．食品としてのレモンは，香りと酸味が重要な香辛料的な果実なのだ.

ビタミンCをとる食品は，現実的に食べやすい性質をもっているものがいいことになる．たとえば，イチゴに含まれるビタミンCは80mg．これを，一皿食べるのは容易なことだ．加熱したジャガイモは，13mgのビタミンCしか含まないが，サラダや煮物などにしてかなりの量を食べられる.

まあ，ビタミンCに限らず，食品に含まれる栄養素は「含有量×食べる量」だということを覚えておきたいものだ.



ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成しています．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★貧乏症が治らない

今まで20MBのHDDを使っていたのに何を血迷ったか，突然，衝動的に180MBを買ってしまった．とっても安かったもので……．60MB×3に分けたものの，20MBの頃の貧乏症が治らず，今でも20MB以内で使っている私は大タコ？　だって，バックアップをとろうとした時，20MB以内なら圧縮されて何とかフロッピー1箱に納まるんだもの．（将来の事は何も考えていないY.Y）

――180MBとは羨ましい．編集部では100MBが最高ですからねぇ．それももう一杯だし，そういえばバックアップをとっている編集者なんて1人もいないなぁ．

★パソコンはかわいい悪魔？

つい先々月，妹が結婚し，その旦那はプログラムで飯を食いたいと，コースに乗っていた農協をやめました．学生時代の友人が畜産農家の後継者育成用教育プログラムを作成中，ダウン，入院とのこと．ああ……パソコンとは人生を変えてしまう悪魔でしょうか．ところで私，給料全部XVIに注ぎ込みました．ああ何と……かわいい悪魔．（道木隆征）

――プログラマーズ・ページ編集部にも畜産農家の後継者でありながら，なぜか編集者になっている人がいます．はたして彼を狂わせたのは，牛かそれともパソコンか……？

★ノート型パソコンになにを求めるか

ノート型パソコンで，ドライブユニットが2基ついたすぐれものが登場してきた今日このごろである．いったいハードウェアの限界はあるのだろうか……？また，ユーザーの求めているものはいったいなんなんだろう……？ただコンパクトにすればいいってものでもないと思うんだけど……？？（渦潮太郎）

――人間の欲望にはきりがありませんから，どこまでいってもユーザーは満足しないんじゃないでしょうか？電子手帳くらいの大きさで，486マシンなみのパワーのあるポケットパソコンが欲しい！

★たまたまハガキを出したら……

7月号のフロッピーディスク到着致しました．創刊号より毎号出していたのですが，6月号だけ出さずにいたのです．というより，6月号から出すのが面倒に（プレゼントも当たらないので）なったのですが，たまたま7月号は出したおかげでDISKが送られてきたのですが，このバックアップ体制は私達，読者には有り難いことです．よって非常に遅ればせながらここに6月号分も送ることにしました．（中川龍一）

――毎号「愛読者カード」を送ってくださるとは律儀な方ですね．ありがとうございます．これを機会に，今後もずーと送っていただければ……うれしい．(^_^;

★管理責任者になったとたん

作業記録の管理にパソコンのデータベースを使わされている職場のものです．メンテする人が誰もいないので「管理責任者」に名乗り出たとたん，貴重なデータが消えてしまう事件発生．気が付くのが遅かったので，5日分は回復せず，約800件再入力してもらいました．バックアップの方法を見直し，反省．誰だ!!　データを消した奴は．（青木与志夫）

――“転ばぬ先のバックアップ”とことわざにもあるとおり，つねにバックアップをおこたりなくやっておくことが大事ですね．とにかく磁気メディアは不安定ですから．

★AT互換機は速い！安い！

ちょうど職場にIBM AT互換機(MITAC製)がきて，DOS/Vが入っています．次号のDOS/Vグラフィックスが楽しみです．日本語ウィンドウも走っています．速い（25MHz)，安い（本体，ハイレゾ，SVGA，40MB HDD，4MB RAMで40万強？）のです．3000US$でi486も購入できるというし，IBMのファンになれそう．（安江正樹）

――秋葉原のコムサテライトでは，486マシンが30万円弱（本体のみ）で売られています．しかし，PC互換機はまだまだ手にいれにくいのが現状です．ショップなどが増えてくるといいのですが．

★OLD-PCの悲しみが……

OSレベルでの標準化の進められている今日この頃，私と

しては嬉しいかぎり．これで対応機種を考えずProgrammingのできる日も夢じゃない．でも，うちのPCたちは，どんどん外界から遠ざかってゆく．これからしばらくの間，OLD-PCの悲しみが中古SHOPに満ちあふれる日が続くでしょう（Win3のせい？）．でも，何が標準のプラットホームになるのでしょうね．個人的にはMacのOSが好き．Win3より絶対に温かいSystemだと思うのですが，どうでしょう．68系とか86系のボーダレスは無理でしょうか？

P.S.うちのVAもDOSマシンなのでWin3を移植してください（8086互換のCPUじゃダメですか？　NECさん）（アドミラルシステムさんの記事を読んでますますModula-2に燃える狂夢）

――最近“標準のプラットホーム”などというしろものは，この世の中に存在しないと考えるようになりました．ようは，自分のプラットホームに何を選ぶかということでしょう．

★C一色でいいのか？

最近，巷はC言語一色で，プログラムはCで書いて当然といった趣さえある．確かにCは優れた言語であるが，R:BASEなどのRDB上の簡易言語でも立派な業務用ソフトを開発できる．非常に細かな処理を実現するために，Cなどを使うことはある．言語とは用途により使い分けるものだと思う．アルゴリズムさえ身に付けていれば，どんな言語にも対応できることを，もう少し認識するべきである．（長谷川勝規）

――表にでてくるのはC一色ですが，実務レベルではRDBも含めて，さまざまな言語が使われているのではないでしょうか．

★みごとな打ち合わせ

先日，わが子（1歳5ケ月）が『カンパイ』といって，私の愛用していたMAGカップとプレゼントでもらったサザンのMAGカップを打ち合わせ（仕様の打ち合わせではなく，字のとおりの）て，みごと2個とも使用不能にしてしまった．（だからMAGカップくださいといいたいのではないけど……）．だれだ，うちの子にこんな悪いことを教えたのは（;_;）．ところで，スピードさえ考えなければ，CよりBASICの方が簡単で良いですね．BASICでヒナ形を作って，Cで機能強化版を作るパターンがわりと良かったりするんですよ．参考までに．（柴田和広）

――BASICでプロトタイピングして，Cで書きおろすわけですね．でも，マイクロソフトのMS-BASIC 7.1ならそのまま最後までいけるんじゃないかな？あのコンパイラは超強力だから……．

★大型汎用機をリセットしてみたい

私は仕事では，大型汎用機を使ってオンラインシステムの構築をやっています．したがってパソコンはあくまで息抜きの趣味の範囲です．パソコンのいいところは，自分の好きなように使えることですね．イヤになったらすぐリセットしてしまいますから．仕事で使っている大型汎用機を一度リセットしてみたいと思っていますが，パソコンのように自由にはならないし……（iCE）

――もうやーめたっ／と，リセットしてしまいたい気持ちはよくわかります．私もできることなら人生をリセットしたいと思うことがあります．

★DOS 5を待とう

今やパソコン界はGUIの時代．MS-WINDOWS 3の快進撃ぶりには驚くほかありません．CHARベースでは殺風景というわけでしょうか．しかし，CHARベースも満更捨てたものでもありません．MS-DOSがVer.5にアップしましたしね．WINDOWS 3を使うとなれば，HDもメモリも大増設は必至．ユーザーの負担は更に重くなります．ここはジッと我慢して，DOS 5を待ちますか．（Ulysses）

――ハイ．私も待ってます．

★本だけじゃ無理みたい

BASIC育ちの私めが何を思ったかCを始めようと思いたち，雑誌を読みあさったり，解説書を買ってみたりしても，いまいち理解できていない今日この頃．ふと本屋で見つけた書誌にCの一文字を見つけて買ってしまった．結局，コンパイラを買わないと，本だけでは無理みたいと今思っています．（ハックル・ベリー）

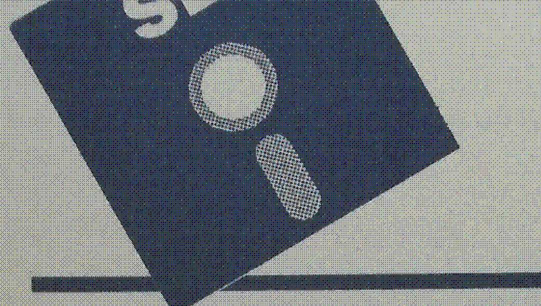
――行くあてがなくても，旅行のパンフレットを見ているだけでワクワクしますよね．たぶん，そんなふうにも感じるんじゃないですか？　早くコンパイラを買って，がんばってくださいね．

★まだ20代ですが……

約10年前は，店先のパソコン（当時はマイコンといったと思う）で，BASICを覚えた．Level 3，80（／），MZ-80などが主流で，FM8，88は夢のかなた．16ビットなどとんでもない／　今や，32ビットマシンが80の値段で買えるようになりつつある．うちの実験室では80がコントローラーとして現役なのになあ．すでに年寄りのような話だけど，私は20代です．（YJO）

――博物館に並ぶにはやっぱり最低100年はたったものでないと．でも，パソコンはたったの10年で，博物館ものになってしまいますからね．

（敬称略）



付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1991年10月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明

ディレクトリ名：ルート

収録ファイル名：

README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）

README.DOC（ディスク内容の説明，MS-DOSテキストファイル）

ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）

ONLINE.DOC（収録されているオンラインソフトの簡単な説明）

V.EXE（ミニページャー，PC-9801専用）

BOOK.TXT（翔泳社書籍目録）

●特集「286/386メモリ活用プログラミング」

③「プログラム実行環境『P386』の作成」（p.28）

ディレクトリ名：TOKU

収録ファイル名：TOKU3.LZH

⑤「DOSエクステンダ活用プログラミング」（p.52）

ディレクトリ名：TOKU

収録ファイル名：TOKU5.LZH

●MS-DOSツール・ラボ（p.82）

Forthコンパイラ「FAST」③

ディレクトリ名：TOOL

収録ファイル名：TOOL.LZH

●実践C塾（p.94）

辞書順ソート

ディレクトリ名：JUKU

収録ファイル名：JUKU.LZH

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名：

EMSFUNC.TXT　EMSファンクションリクエスト一覧（MS-DOSテキストファイル）

●オンラインソフト採集帳（p.136）

今月号のオンラインソフト採集帳で紹介したNEC独自文字をJIS文字に変換するフィルタ「DENEC」，特集1で紹介されたメモリ使用状況表示プログラム「MS Ver.3.11」が収録されています．また，LZHという拡張子を持つ圧縮ファイルを復元するのに必要な，高圧縮復元書庫管理ソフト「LHA」（本誌4月号参照）も収録しています．

ディレクトリ名：ONLINE

収録ファイル名：

MS311.LZH（MS Ver.3.11）

DENEC.LZH(DENEC，S-JIS採用MS-DOSマシン)

LHA213.EXE(LHA Ver.2.13，MS-DOS汎用)

●クイズ・ランダムスクリーン（p.142）

クイズ解読ツール「PUZLE」

本誌掲載のクイズランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ替えることにより，解答を得ることができるユーティリティです．PC-9801シリーズです，詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE

収録ファイル名：

PUZLE.EXE（解読ツール，PC-9801専用）

PUZLE.DOC（PUZLEの説明）

CONFIG.SYS（PUZLE用CONFIG.SYS）

AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）

RNDSCREN.LZH（画像データ）

PETER.LZH（9月号の解答データ）

PETER.BMP（Windows 3.0の壁紙）

※使用上の注意

本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

C平　ひさしぶりにアメリカ製のゲームをやったら，VGAを対象に作られたゲームは，絵の質がすごくよくなっている．

P子　ひさしぶりって，ソリティアとかマインスイーパーというのは，ゲームじゃないわけ．

C平　あれは，息抜き程度のものじゃない．いまいっているのは，アドベンチャーゲームみたいにまとまった時間をとってやるものなの．VGAのゲームは，320×200で256色のモードを使っているんだけど，去年あたりからそれを基準にしてゲームが作られるようになってきたんだ．

P子　たしかに，数字としては16と256ではかなりの違いだけど……．

C平　表現力がすごいの．なんといってもアドベンチャーものがリアルになった．16色だと平面的で色も原色に近いからドラエモンのアニメみたいじゃない．それが，ゴルゴ13の劇画タッチになった感じ．

P子　なるほど，ドラエモンとゴルゴね．それで，ゲームはどんななの．

C平　「HEART OF CHINA」というのをやったんだけど，雰囲気としてはインディ・ジョーンズの「魔宮の伝説」．1930年代のアジアを舞台に誘拐された令嬢を助け出すという物語なんだけど，忍者もでるし，飛行機も戦車もあってなかなか楽しめる．

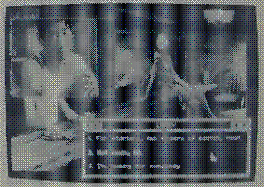
P子　このごろ寝不足の顔してるのはそのせいなのね．

C平　でも，サウンドボードがないから，いまいち迫力が……．サウンドボードが欲しいヨー．

■HEART OF CHINA

シエラオンラインジャパン㈱
動作条件(256色版)：VGA，80286以上，640KB，HDD

誘拐された富豪の令嬢を求めて主人公ラッキーが1930年代のアジアを舞台に活躍するアドベンチャーゲーム．香港にはじまる物語は，成都に，さらにカトマンズ，イスタンブール，パリにと，ユーラシア大陸を横断する大ロマンになっている．
実写の画像と手描きの絵を組み合わせた技法は，制作にあたったDYNAMIXの独特のもの．操作は，すべてマウスでできる．



読者プレゼント

1 LSI C-86

1名
提供：エル・エス・アイジャパン㈱

2 HEART OF CHINA*
(IBM PC用)

3名
提供：シエラオンラインジャパン㈱

3 まいとーく Ver.2
(PC-9800用，5"2HD)

2名
提供：㈱インターソフト

4 LAPLINK-J
(PC-9800，J-3100用)

1名
提供：㈱インターソフト

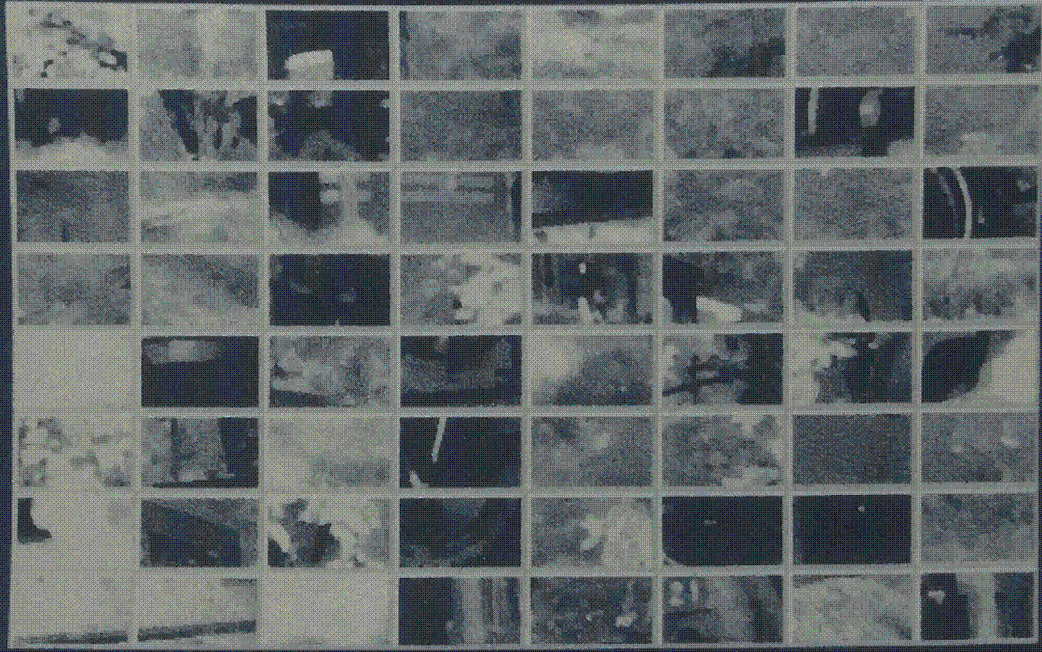
応募は本誌に添付のアンケートはがきをご利用ください．(1991年10月17日消印有効)．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

＊HEART OF CHINAをご希望の方は，ご使用のビデオシステム(EGAかVGA)とメディアサイズ(3.5"か5")を明記してください．

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：石炭で走る乗物，シュッポッポ．



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

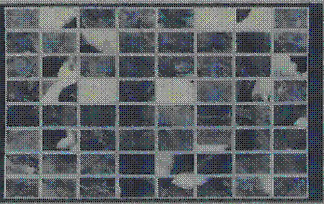
締め切り：1991年10月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

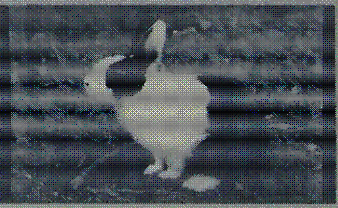
＊画像データは，付録ディスクやBBSでもサービスしています．

9月号の問題と正解

【問題】



【正解：うさぎ】



月刊プログラマーズ・ページ オンラインサービス（SE-NET）のお知らせ

SE-NETでは，創刊号からの掲載プログラムとオンラインソフトのサービスを行っています．また，「ドラゴン・ポスト」や「編集部への質問」，「プログラマ内輪話」などのコーナーも設けております．読者のみなさまと編集部とのコミュニケーションの場としてもご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

①接続するとIDの入力を求めてきますので，GUESTと入力しリターンキーを押してください．
②お知らせを表示後，メインメニューのコマンドプロンプトが表示されます．
③月刊プログラマーズ・ページへのご意見などは「ドラゴンポスト」ボードに書き込んでください．
④ファイルの転送が終了したら，0と入力して回線を切ります．

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ バックナンバーのご案内

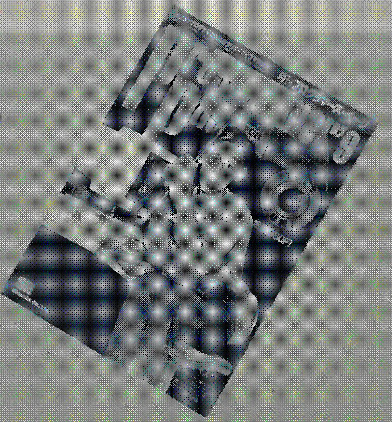
■定価・各980円

（1991年1月号「特集・プログラマの仕事場」・2月号「特集・Windowsの時代である!」・3月号「特集・この言語を使いなさい!!」・4月号「特集・C読本」は品切れです）

1991年5月号 5インチFD付き

特集・基礎力としてのアセンブラ

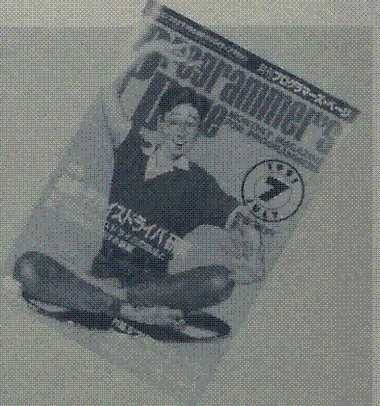
プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ．コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす．



1991年6月号 5インチFD付き

特集・研究「プログラミング統合環境」

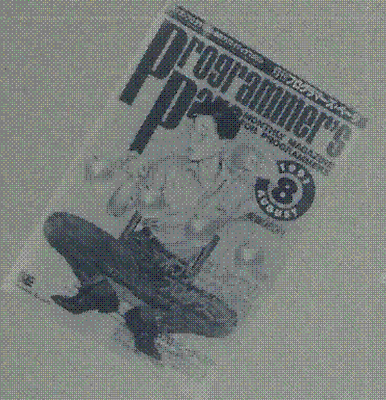
今流行りのエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する．



1991年7月号 5インチFD付き

特集・最新「デバイスドライバ」研究

パソコンの拡張性を支えるソフトウェアインターフェースであるデバイスドライバ作成の技法と特徴をMS-DOS，Windows，OS/2の各環境ごとに解説する．



1991年8月号 5インチFD付き

特集・はじめての オブジェクト指向プログラミング

90年代のプログラミングパラダイムとして注目される“オブジェクト指向”の歴史的な発展の過程から基礎概念，分析・設計の方法までを網羅的に解説する．

1991年9月号 5インチFD付き

特集・グラフィックス プログラミングの世界

グラフィックスプログラミングの基礎からフレームバッファの活用，レイトレーシング，Windows，IBM PCのグラフィックスなどその技法を徹底的に解説する．

「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで1か月ほどかかります）．

小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を下記までお送りください．5，6，8，9月号の送料は1部81円です（7月号は1部86円）．2冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください．

〒102 東京都千代田区平河町2-4-14
平河町KSビル4階
(株) 翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ11月号
(10月18日発売)

付録:5″ 2HDフロッピーディスク付き!!

特集
Windowsプログラムの作り方

Windows 3.0が鮮烈なデビューを飾ってから1年以上がすぎた．貧弱といわれたWindowsプログラム開発環境も，サードパーティーから発売された多くのツールによって様変わりをはじめている．正統派のMS-C+SDKをはじめVisualBASIC，BC++，TPWなど，次々に登場した新しい開発環境によるWindowsプログラムの作り方を探る．

月刊プログラマーズ・ページ　1991年10月号
1991年10月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋佐藤裕美＋小林弘人(図版デザイン)＋友田圭治
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

© 1991 SHOEISHA
本書の一部または全部を著作権法の定める範囲をこえ，無断で複写，転載，テープ化，データファイル化することを禁じます．
＊本書の内容に関するご質問は，ご面倒でも必ず文書にて編集部までお問い合わせください．

EDITOR'S PAGE

●IBM PC10周年だそうだ．「ニューヨークシステム通信」佐々木氏の原稿を見て，なるほどと思った．ということは，DOSも10周年，しかし，世界中で数千万本売れたというOSの，10周年行事があるという話は聞かない．これに限らず，パソコンの世界では，○○周年という言葉はどうもぴんとこない．回転が非常に速いこの世界では，年単位の話はもう感覚的に馴染まなくなっているのだろうか．ふと考えてしまった．(TA)

●先月の編集後記に書いたようにDOS 5.0+DOS/Vという組み合わせでDOSを使用しているが，やはり多少問題があるようだ．たとえば，日本語Windowsのメイン.GRPのようにファイル名に日本語を使用していると動作がおかしくなる．この場合はWindowsのメイングループが作成されなくなってしまう．MAIN.GRPとリネームし，PROGMAN.INI中の［Groups］の指定をこれに合わせて書き換えれば，一応動くようになるが，あまり気分のいいものではない．早く正式な日本語DOS 5.0が欲しい／(嘉平)

●夏休みに佐渡の隣にある「粟島」へ行ってきました．約3時間かけて新潟から，電車→タクシー→船と乗り継ぎ，ようやく辿り着いた私を待っていたのは，きれいな海と新鮮な魚．都会ずれした私もつい「こんな島で暮らしたいな……」とつぶやいていました．それを聞いていた民宿のおばちゃん「たまに来るからそうゆうこというのよ」．たしかにすることなくて8時には寝てましたが．(ま)

●「ペットを飼いたい」そう思いつつも狭い1DK，しかも台所がときに暗室となるわが家ではとても無理だと思っていたら，いいものが見つかりました．タカラの愛鳥倶楽部，ほぼ原寸サイズの小鳥の置物で，振動に反応してピィピィ鳴くのです．テーブルの上にコップをおけばピィピィ，窓をあけて風が入ってくるとピィピィ．あんまり可愛らしいので会社に持っていこうと思ったけれど，よく考えたら会社まで持っていく間にピィピィやられたのではかなわない．会社用にもう1羽飼おうかどうしようかとマジメに考えているこのごろです．(kos)

●すでに日本の夏は終っていた．
でもこの本が発売される頃はハワイで優雅な夏を過ごしているだろう．(Yoshi)

お詫びと訂正

●本誌9月号130ページで，「CD-RTOS/RAVE」の発売元の社名がまちがっていました．正しくは「マイクロウェア・システムズ（株）」です．訂正するとともに，ご迷惑をおかけした方々にお詫びいたします．

●連載「プログラム「遊」工房」は，著者急病のため休載いたします．

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由	パソコンの所有形態(機種名)
□1.書店店頭　□2.他社出版物(　　)	□1.個人所有　(　　)
□3.小社出版物　□4. DM　□5.口コミ	□2.職場にある(　　)
本誌を購入した書店　市(区)　書店	□3.持っていない
よく使用するソフトウェア	よく使用するOS
今月号でおもしろかった記事(いくつでも)	貴方が選ぶベスト3
	パソコン雑誌 ①　②　③
今後取り上げてもらいたい特集のテーマ	ソフトウェア ①　②　③
	パソコン本体 ①　②　③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

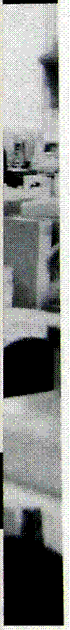
業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業　④建設業　⑤通信・運輸　⑥システム・ソフト　⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者　④プログラマ　⑤その他の技術者　⑥専門職　⑦事務・営業職　⑧経営管理職　⑨商工サービス　⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

希望項目 / 広告資料請求No.												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C



ソース
たシス
を
プレッ
CAD
発・販売
すめて
針にあ

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払
麹町局承認
3908
差出有効期間
平成４年11月
30日まで
切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ
READER'S SURVEY CARD - 愛読者カード

氏名	性別	生年月日
	□1男 □2女	19　年　月　日

住所　〒	電話
	（　　）　－

職業	ユーザー区分
□1ソフト技術者 □2ハード技術者 □3技術系会社員	□1プロ
□4事務系会社員 □5営業系会社員 □6教師	□2マニア
□7一般公務員 □8経営管理職 □9理系学生	□3一般ユーザー
□10文系学生 □11その他（　　）	

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払
麹町局承認
3910
差出有効期間
平成４年11月
30日まで
切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓　　名	5	生年月日	19　年　月　日

6	自宅住所	〒　電話　（　　）	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話　（　　）	

払込票
通常払込料金加入者負担
口座番号 東京 4-15493
加入者名 株式会社 翔泳社
金額 11760
払込人住所氏名 ※
備考
受付局日附印
切り取らないで郵便局にお出しください。
記載事項を訂正した場合は、その箇所に訂正印を押してください。
払込通知票
通常払込料金加入者負担
口座番号 東京 4-15493
加入者名 株式会社 翔泳社
金額 11760
料金 払込み 特殊 円
備考
受付局日附印
払込人住所氏名 ※（郵便番号 ） TEL（ ）
この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）
各票の※印欄は、払込人において記載してください。

大部分を受注系の大・中規模ン（SI）サービスが占めていューザーにはなじみがうすいソフト「ユーカラ」や「BUSI社」といえば思いあたる人も年に㈱東海クリエイトの社名更した。現在では、本社のあど全国７か所に事業所を構え、トウェアハウスである。

社総務部部長土屋氏、パソコ氏、同部商品開発部部長安田話をお伺いした。

得意分野をつくらないのが特徴

クレオのおもな事業は、システムインテグレーション（SI）サービス、パッケージソフトウェアの開発・販売の２つで、このうち受注系の大・中規模のSIサービスが85～90％を占めている。「得意分野をつくらないのが特徴」というように、クレオのソフトウェア開発は非常に多分野に及んでいる。一般企業向けアプリケーションシ事業向けシステム、通信システム、メーカー系のOS開発など、数えだしたらきりがない。その多くが何百人月というおおがかりなもので、ひとつのプロジェクトに常に何十人という人間が携わっているのが特徴だ。

〝くるものは拒まず〟的なチャレンジ精神で、これまで新しい分野をどんどん開拓してきたので、社内には貴重なリソースがたくさん蓄えられている。現在同社が新しめていることは、それらのリソースを活かして、パッケージを中心としたシステム販売を推進していくことだ。

すべての核となるような仕事を

パソコンの分野では、これまでスプレッドシート、ワープロ、データベース、CADなど数多いパッケージソフトの開発・販売をパソコン事業部独自の方針ですすめてきた。しかし今後は、会社全体の方針にあ

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。　（郵　政　省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

ーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

6	自宅住所	〒　電話（　）	○資料送付希望先の番号に印をつけてください。
7	会社住所	〒　電話（　）	

□7一般公務員　□8経営管理職　□9理系学生　□3一般ユーザー
□10文系学生　□11その他（　）

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

綴じ込みの
1

ちょっと気になるこの会社

株式会社クレオ

クレオは、その業務内容の大部分を受注系の大・中規模のシステムインテグレーション（SI）サービスが占めている。というと、一見パソコンユーザーにはなじみがうすいように思われるが、〝ワープロソフト「ユーカラ」や「BUSI COMPO」をだしている会社〟といえば思いあたる人も多いだろう。クレオは、1974年に㈱東海クリエイトの社名で設立、1989年に現社名に変更した。現在では、本社のある東京のほか、東北、関西など全国７か所に事業所を構え、社員数450名を誇る大手ソフトウェアハウスである。

港区芝浦の本社を訪ね、同社総務部部長土屋氏、パソコン事業部計画管理部部長大森氏、同部商品開発部部長安田氏に、採用・開発に関するお話をお伺いした。

得意分野をつくらないのが特徴

クレオのおもな事業は、システムインテグレーション（SI）サービス、パッケージソフトウェアの開発・販売の２つで、このうち受注系の大・中規模のSIサービスが85〜90%を占めている。「得意分野をつくらないのが特徴」というように、クレオのソフトウェア開発は非常に多分野に及んでいる。一般企業向けアプリケーションシステム、金融・保険業向けシステム、公共事業向けシステム、通信システム、メーカー系のOS開発など、数えだしたらきりがない。その多くが何百人月というおおがかりなもので、ひとつのプロジェクトに常に何十人という人間が携わっているのが特徴だ。

〝くるものは拒まず〟的なチャレンジ精神で、これまで新しい分野をどんどん開拓してきたので、社内には貴重なリソースがたくさん蓄えられている。現在同社が新しくはじめていることは、それらのリソースを活かして、パッケージを中心としたシステム販売を推進していくことだ。

すべての核となるような仕事を

パソコンの分野では、これまでスプレッドシート、ワープロ、データベース、CADなど数多いパッケージソフトの開発・販売をパソコン事業部独自の方針ですすめてきた。しかし今後は、会社全体の方針にあ

わせた商品というものを開発していきたいという。パッケージ開発やシステムインテグレーションで培った技術を統合し、パソコンからホストまで、すべての核となるようなソフト製品をつくる部門を組織化したいとの意向だ。これにより、顧客へのサポートがいっそう強化されると同社は考えている。

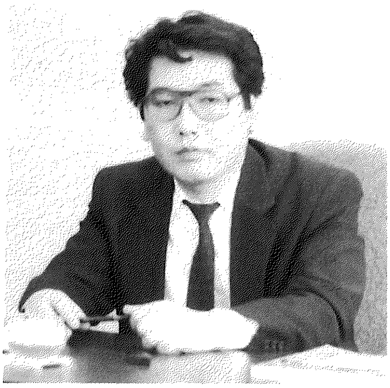
最近、パッケージでは、教育市場向けに言語製品「True BASIC」をだしたり、MS－Windows関連ソフトの開発なども増えてきている。また、顧客にはコーポレイト（法人）ユーザーも多いので、今後もその分野向けのアプリケーション開発に重点をおくという。

技術的なものよりヤル気を重視

現在の社員数は約450名、平均年齢26.5歳。とにかく若いという印象を受ける。今年の採用実績は新卒・中途あわせて約100名で、今後2、3年はこの規模で採用していく予定。この数からみても、クレオが現在急成長中のソフトハウスであることがわかるだろう。新卒の内訳をみると、コンピュータ専門学校、4年制大学などで、4大では文系が8割前後を占めるという。

「いちおう全員に適性試験をおこないます。けれど、SEをやりたいという人はSEをやる資格があると考えているので、技術的なバックグラウンドはそれほど重視していません」（採用担当土屋氏）というように、中途の場合でも必ずしも経験者でなければならないということはないそうだ。

入社後半年間を研修期間とし、3か月を講義、3か月をOJTとしている。配属に関しては、「SEとひとくちにいっても、パッケージ開発部門では商品企画担当、開発主体の人、ユーザーサービス担当、受注部門ではフィールドSE、テクニカルSE、コンサルタントSEなど、非常に幅が広いんです。入社時にはそういった区分のみ認識してもらって、配属は人事のほうでほぼ決めます。そこで2、3年経験を積んだ時点で、各自が自分の適性を考えて選ぶというようにしています」。



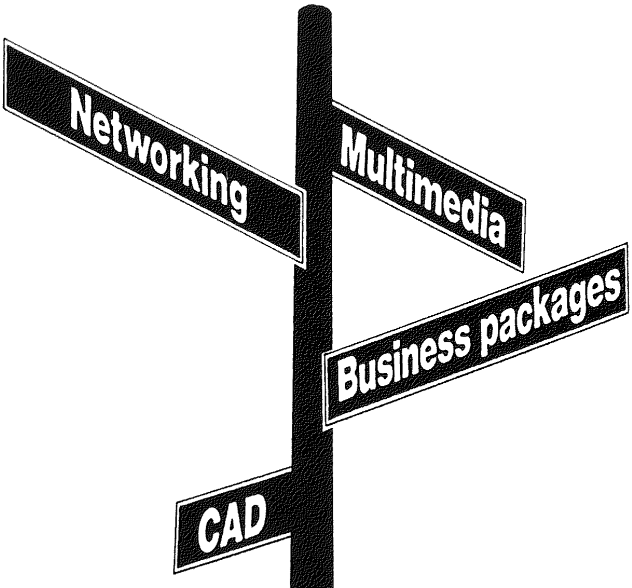
写真上から、パソコン事業部商品開発部部長安田知弘氏、パソコン事業部計画管理部部長大森俊樹氏、総務部部長土屋淳一氏。

オタクっぽいだけでは……

また、求人像については、「オタクっぽいだけの人はちょっと困りますね。うちの会社は顧客と接することが多いので、ビジネスセンスが要求されます。とくに大・中規模の開発部門では、何百人月という大がかりなプロジェクトに携わるわけですから、プロジェクト管理のできる人が望まれます」。読者諸氏の中には耳の痛い人もいるかもしれないが、これは最近のソフトハウスの望むSE像の特徴といえるだろう。個人ユーザーの趣味としてのソフトをつくる時代は終わり、企業の業務としてのソフト開発が中心の現在、SEに望まれるものは、より広い視野と知識、顧客とのコミュニケーション能力などである。

そうはいっても、パソコンパッケージの開発が中心の同社の子会社㈱クレオアールアンドディには、オタクっぽい人もそれなりにいるとのこと。幅広い器のある会社だけに、自分の適性にあった場所をさがすことができそうだ。

クレオが最近力をいれているパッケージソフトが、教育市場をターゲットにしたという「True BASIC」。 これは、構造化プログラミングの概念が優しく理解でき、基礎力が自然と身につくというBASIC言語。 初心者は一度トライしてみてはいかが？



高水準の技術開発にはパーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

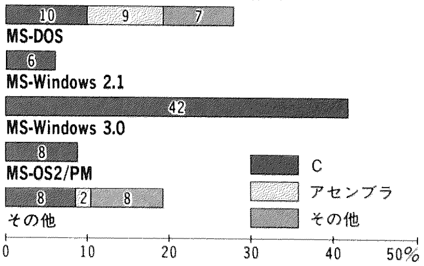
会社概要　ELTEX

設　立	昭和60年12月14日
資本金	1,000万円
事業内容	パソコンを中心としたシステムの提案・開発・販売　①CD-ROM・LD　など新しいメディアに対応したシステムの開発 ②ネットワークの端末システムの開発 ③特定業種向けCADの開発・販売 ④基本ソフトの開発
特　徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している。また、ＣＡＤを中心とした自社プロダクツにも積極的で、その比率も増加している。基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、ＯＥＭ製品のＢＩＯＳ・ユーティリティなどを企画から手掛けている。
取引先	三菱電機、三菱自動車、日本航空　他
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募集要項　ELTEX

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上　20～36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）～46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理１種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理１種・住宅手当含む）
手　当	時間外手当、役職手当（5000円～６万円）、住宅手当（１万5000円）、資格手当（２種：5000～特種：２万円）、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月）、決算賞与年１回（平成１年度実績１ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00～16:00）試用期間中は標準時間9:00～17:45
休日休暇	完全週休２日制（土・日）、祝日、年３回連休（GW・夏期・年末年始）、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書（写真貼付）、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045)322-9667　総務課

最近のOS・言語別開発動向



採用HOT・LINE
TEL.(045)322-9667

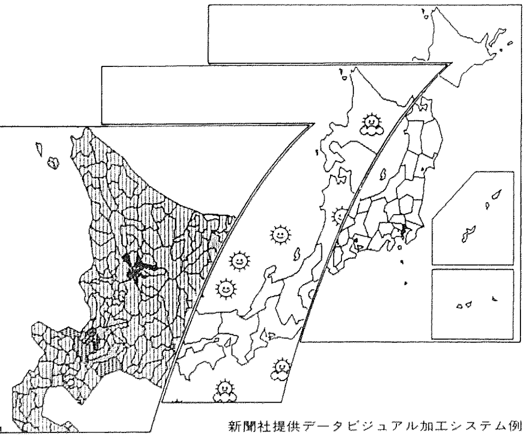
株式会社エルテックス

本当の技術者になってください。

随分昔のことになるかもしれません。初めて組んだ自分だけのシステム。それを動かすときの緊張、そして、動いたときの満足度。何にも変え難いものがあるに違いありません。その時のあなたは本当の技術者でした。自分で開発したものを自分で動かすときの喜びは、技術者の特権でした。今はどうでしょうか。システムの一部分を開発するだけで、全体のシステムを把握することもできない。完成品を自分で動かすこともできないでいる。そんな技術者を、本当の技術者と呼べるのでしょうか。バーテックシステムでは、3・4名のチーム体制を組んで開発していくから、皆が皆責任を持って協力しあいます。そして、自分たちで開発したシステムを動かすときの喜びは、当時のそれと変わっていません。納期（スパンはだいたい4カ月～10カ月）までに高度なシステムを開発する。この技術者としてのモラルさえ守ってくれれば、それ以外の束縛でメンバーを苦しめたくありません。

本当の技術者が、バーテックシステムにはいます。

技術者随時募集



新聞社提供データビジュアル加工システム例

開発分野	主な開発実績
基本ソフト開発	金融端末用基本ソフト開発 POS端末用BIOS開発 プリペイド・カード　ドライバ開発
言語ソフト開発	ハンデー・ターミナル用BASICインタプリンタ開発 POS端末用BASICデバッカ・プログラム開発
パソコン・ソフト開発	大手新聞社の端末側システム開発 パソコン・株価情報検索システム開発 パソコン・パッケージソフト移植 NAPLIPS仕様ソフト開発 予算管理プログラム開発 フォント作成ユーティリティ開発 音声チューニングシステム移植
開発言語	C、アセンブラ、他（Windows）
使用機種	PC-9800シリーズ、J3100、IBM5550、FM-R

募集要項

職種●ＳＥ・プログラマー（正社員）

資格●高卒以上　①未経験者18～27歳迄　②経験者（6ヶ月以上）20～30歳位迄　※Ｃ言語、アセンブラの経験者、特に歓迎。

給与●①15万円以上　22歳／16万円以上　②17～35万円　22歳／18万円以上　30歳（経験5年例）／30万円　※経験・年齢・前給を考慮し、面談の上、相談に応じます。

待遇●昇給年1回（あなたの努力によって特別賞与あり）、賞与年2回、交通費全額支給、各種社会保険完備

勤務地●本社（池袋）

勤務時間●9:00～17:30　フレックスタイム制（コアタイム10:00～16:00）※出社、退社はあなたの自己管理にお任せします。

休日休暇●完全週休2日制（土・日）祝日、年末年始、夏季、有給休暇（初年度6日、次年度8日、以降毎年1日ずつ加算します）　※年間休日122日＋有給休暇

応募●電話連絡の上、履歴書（写真貼付）を持参下さい。都合の悪い方、郵送でも結構です。※今すぐ転職できない方、入社日応相談。

交通●ＪＲ・地下鉄、池袋駅東口徒歩8分

会社概要

設立●昭和59年5月18日

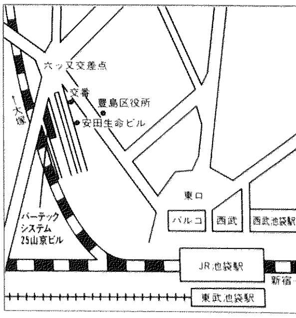
資本金●500万円

従業員数●9名

事業内容●アセンブラ、Ｃ言語によるパソコン開発。メーカー系基本ソフトの開発。新聞社提供データビジュアル加工パッケージ等の開発他。3・4名のメンバー体制。

TEL 03-3986-9435……………………人事担当

〒170 豊島区東池袋1-48-10　25山京ビル4階



バーテックシステム株式会社

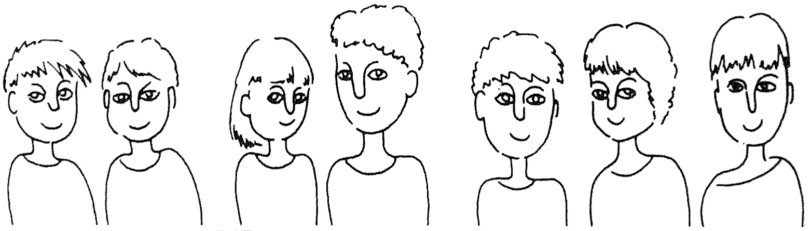
ロジック・アーツ株式会社

これから何をやろうか

「高品質な製品を効率よく開発する」
設立当時(4年前)に掲げた我が社のテーマ。
そして夢は一歩近づいた。
協力しあう技術者集団と広義に渡る開発環境が
「夢実現」への近道となった。
後はどれだけ実績を積むか、
どれだけ組織の「和」を広げられるかにかかっている。
そのためには君たちの参加が必要となってきた。
君たちが我が社に訪れたとき
君たちの夢を聞きたい。
そして一緒に
「これから何をやろうか」考えていきたい。

事業内容	基本ソフトの開発からパソコン・マイコンの応用システムの企画・開発・販売、電子回路の設計及び製造まで幅広く手掛けています。
職　種	SE・プログラマー ※派遣は一切ありません。
資　格	高卒以上　20歳～30歳位
給　与	月給　22万円以上　経験・年齢考慮の上、当社規定により決定。
勤務時間	9：30～18：00
休日休暇	完全週休2日制(土・日)、祝日、夏期・年末年始・有給・特別・慶弔休暇
待　遇	昇給年1回、賞与年2回、交通費全額支給、各種社会保険完備
応　募	電話連絡の上、履歴書持参ください。面接日・入社日応相談
交　通	有楽町線・東武東上線志木駅徒歩7分
言語・OS	CP/M、MS-DOS、UNIX、アセンブリ言語(Z80、64180、8086、68000)、C言語(hitech-c、ms-c、TURBO-c、etc.) ※HP-IB、RS-232cで市販計測器または特注製作品とパソコンによる自動計測システムには数多くの経験があります。近年ではワークステーションによる画像処理も手掛けています。)

〒352　埼玉県新座市東北2-13-25 城取ビル2F
TEL.0484(74)2225 担当/鈴木



技術者随時募集

人材募集広告出稿のご案内

『月刊Programmer's Page』では求人広告を取り扱っています。

プログラマを必要とするなら
「Programmer's Page」の読者をおすすめします。

- ●理由 その1　「Programmer's Page」の読者は全員プログラマ。
- ●理由 その2　「Programmer's Page」の読者にはプロの技術者が多い。
- ●理由 その3　そしてキャリア3～5年、20代の若い読者が多い。

■広告料金

スペース	普通サイズ	ブリードサイズ
4色1頁	350,000円	380,000円
2色1頁	250,000円	280,000円
1色1頁	200,000円	220,000円
1色1/2頁	150,000円	使用できません

■上記の料金に制作料は含まれていません。

■単独差し込みハガキ………40万円

■広告原稿寸法

■見開
普通サイズ…………天地230mm × 左右340mm
ブリードサイズ…………天地257mm × 左右364mm

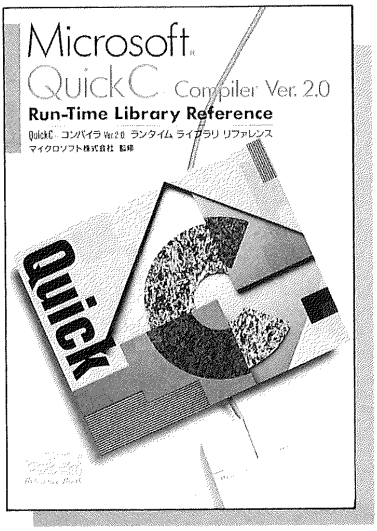
■1頁
普通サイズ…………天地230mm × 左右160mm
ブリードサイズ…………天地257mm × 左右182mm

■1/2頁
普通サイズ…………天地110mm × 左右160mm
ブリードサイズ…………使用できません

SE Shoeisha Co., Ltd.
詳しいお問い合わせは
『03-3263-0447』株式会社翔泳社 マーケティング部広告企画課

プログラミングのための優良図書

SE
Shoeisha Co., Ltd.



Quick C コンパイラ Ver.2.0 ランタイムライブラリリファレンス
Quick C Compiler Ver.2.0 Run-Time Library Reference

マイクロソフト株式会社　監修
A5判/364頁
定価2,400円

『言語ソフトのベストセラー「Microsoft Quick C」のオンラインヘルプを完全書籍化。マイクロソフト社監修によるパーフェクトQuick C リファレンス。プログラミング時の参考書として、また学習書として活用できます。』

Quick BASIC Ver.4.5 ステートメント&関数リファレンス
Quick BASIC Ver.4.5 Statement & Function Reference

マイクロソフト株式会社　監修
A5判/394頁
定価2,400円

『Quick BASIC Ver.4.5のQBアドバイザを完全書籍化。パソコンの前を離れて参照したい時など学習用として最適です。各ステートメント & 関数にサンプルプログラミングを満載したパーフェクトリファレンス。マイクロソフト社監修。』

Quick BASIC スタイルブック

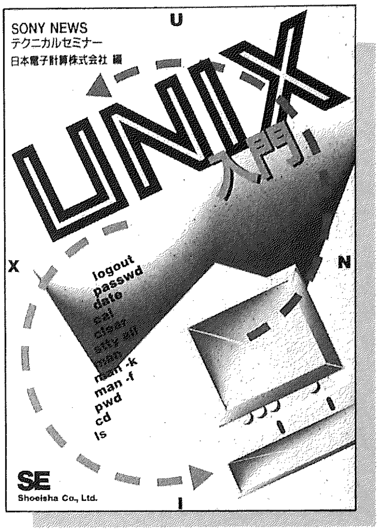
イント・ツーワン　著
B5判変型/236頁
定価2,300円

『初心者からマニアまで幅広く使われる Quick BASIC。この Quick BASIC を用いて質の高いプログラミングを書くためのノウハウがいっぱいつまっています。一問一答の問題集形式。パズルのようにプログラミングが学べます。』

UNIX入門
―SONY NEWSテクニカルセミナー―

日本電子計算株式会社　編
A5判/237頁
定価1,800円

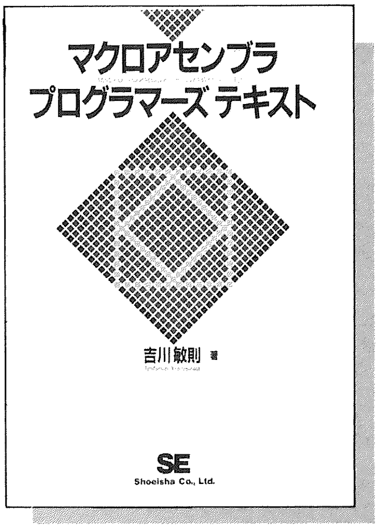
『ソニーのNEWSによる「UNIX入門」セミナーのテキストをベースとした入門書。UNIXのコマンド、エディタの使い方、日本語入力の方法、ネットワークの利用方法などを豊富な実例をまじえて解説。はじめてのワークステーションを利用するユーザ必携の書。』



マクロアセンブラ プログラマーズテキスト

吉川敏則　著
A5判/243頁
定価2,600円

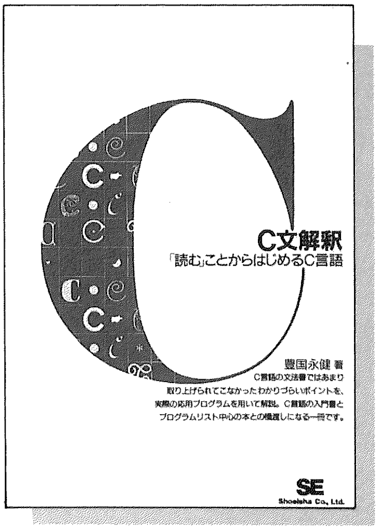
『最低限必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説。アセンブリ言語に興味を持つ人や、これからアセンブリ言語でプログラムを作成する人に役立つ一冊。』



C文解釈
―「読む」ことからはじめるC言語―

豊国永健　著
B5判/183頁
定価2,000円

『C言語の文法書ではあまり取り上げられてこなかったわかりづらいポイントを、実際の応用プログラムを用いて解説。C言語の入門書とプログラムリスト中心の本との橋渡しになります。』



SE Shoeisha Co., Ltd. 株式会社翔泳社 〒102東京都千代田区平河町2−4−14平河町KSビル TEL.03−3263−0447(代表)

翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

米国「The C Users Journal」「TECH Specialist」提携誌

C JOURNAL

C言語を主体とするコンピュータプログラマのための月刊情報誌
月刊 Cジャーナル

10 OCT. 1991

定価980円(本体951円)

10月号 9月8日発売

特集 マルチメディア

マルチメディアのためのプログラミング
概説マルチメディア

付録DISK(5"2HD)

- ●CUG PDS#343 Cイメージプロセッシングシステム
- ●CUG PDS#345 TCL/TLP
- ●simkey最新版
- ●掲載記事の全コード収録

●連載●

- ★STANDARD C
- ★Dr. C's Pointers
- ★Questions & Answers
- ★Programming Style 第3回
- ★MacでSmalltalk 第9回
- ★BART EXPRESS 第14回
- ★たのむぜ／DOS/V／ その3
- ★グラフィックユーザインタフェイス環境 第3回
- ★MacのクラスライブラリとGUI 3
- ★[新連載] Pascal言語系の制作 第1回

11月号 10月8日発売 特集 ウィンドウプログラミング

株式会社ビレッジセンター 出版局
〒101 東京都千代田区三崎町3－1－16 神田アメレックスビル2F
●営業部:03-3221-1825●編集部:03-3239-1950

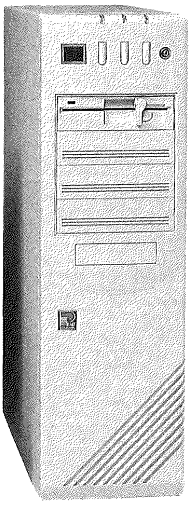
C JOURNALは毎月8日全国の書店でお求めになれます。

Polywell 80486DX Power IBM PC Compatible System

80486DX-25 or 33
AMI BIOS
128K Cashe
RAM Up to 32MB on Motherboard
Full Size Tower
220 Watt Power Supply
Western Degital IDE HD/FD Controller
Maxter HD(200MB)
TEAC 3.5"FDD*1
TEAC 5"FDD*1
2Serial/1Parallel I/F
Orchid ProII Super VGA Card(512K)

Keytoronic 101 Keyboard
MicroSoft Mouse
MS-DOS 4.1
MS-Windows 3.0

HITACHI 14" Super VGA Display
(1024*768 non-interlaced)
初期不良無償交換。本体1年間保証。



25Mhz		40MB HD	80MB HD	120MB HD	200MB HD
	1MB RAM	¥559,000	¥584,000	¥600,000	¥656,000
	2MB RAM	¥573,000	¥598,000	¥614,000	¥670,000
	4MB RAM	¥589,000	¥614,000	¥630,000	¥686,000

33Mhz		40MB HD	80MB HD	120MB HD	200MB HD
	1MB RAM	¥600,000	¥625,000	¥641,000	¥697,000
	2MB RAM	¥614,000	¥639,000	¥655,000	¥711,000
	4MB RAM	¥630,000	¥655,000	¥671,000	¥727,000

米国The C Users Group日本支部
VILLAGE CENTER INC.

■本部：〒101 東京都千代田区三崎町3-1-16神田アメレックスビル2F
TEL.03-3239-2187 FAX.03-3221-1768
■Shop：〒101 東京都千代田区神田神保町1-35-15 だるまビル1F
TEL.03-3219-6113

スタッフ募集中!!▶明るくヤル気のある方大歓迎です。詳しくは電話3239-2187／担当:中村までお問い合わせください。

EMSファンクションリクエスト(INT 67H)一覧(PC-9800シリーズ対応)

■基本ファンクション

●01　Get Status(ステータスの取得)
メモリマネージャが存在し,ハードウェアが正しく動作しているかを取得する
(呼)AH←40H
(返)AH←00H　正常実行
　80H,81H,84H　エラー

●02　Get Page Frame Address(ページフレームアドレスの取得)
ページフレームが位置するセグメントアドレスを取得する
(呼)AH←41H
(返)AH←00H　正常実行
　80H,81H,84H　エラー
　BX←ページフレームセグメントアドレス

●03　Get Unallocated Page Count(未アロケートページカウントの取得)
未アロケートページ数と,拡張メモリのページ総数を取得する
(呼)AH←42H
(返)AH←00H　正常実行
　80H,81H,84H　エラー
　BX←未アロケートページ数
　DX←総ページ数

●04　Allocate Pages(ページのアロケート)
プログラムが要求するページ数をアロケートし,そのページにEMMハンドルを割り付ける
(呼)AH←43H
　BX←アロケートしたいページ数
(返)AH←00H　正常実行
　80H,81H,84H,85H,87H,88H,89H　エラー
　DX←EMMハンドル

●05　Map/Unmap Handle Page(ハンドルページのマップ/アンマップ)
EMSメモリ内のマッピング可能な領域にある物理ページに論理ページをマッピングする
(呼)AH←44H
　AL←物理ページ番号
　BX←論理ページ番号
　DX←EMMハンドル
(返)AH←00H　正常実行
　80H,81H,83H,84H,8AH,8BH　エラー

●06　Deallocate Pages(ページの解放)
EMMハンドルにアロケートされている論理ページを解放する
(呼)AH←45H
　DX←EMMハンドル
(返)AH←00H　正常実行
　80H,81H,83H,84H,86H　エラー

●07　Get Version(バージョンの取得)
EMSインターフェースのバージョン番号を取得する
(呼)AH←46H
(返)AH←00H　正常実行
　80H,81H,84H　エラー
　AL←バージョン番号

■拡張ファンクション

●08　Save Page Map(ページマップのセーブ)
ページマッピングレジスタの内容をセーブエリアにセーブする
(呼)AH←47H
　DX←EMMハンドル
(返)AH←00H　正常実行
　80H,81H,83H,84H,8CH,8DH　エラー

●09　Restore Page Map(ページマップのリストア)
セーブエリアからページマッピングレジスタの内容をリストアする
(呼)AH←48H
　DX←EMMハンドル
(返)AH←00H　正常実行
　80H,81H,83H,84H,8EH　エラー

●12　Get Handle Count(ハンドルカウントの取得)
OSのハンドル0を含むオープンしているEMMハンドルの数を取得する
(呼)AH←4BH
(返)AH←00H　正常実行
　80H,81H,84H　エラー
　BX←オープンしているEMMハンドル数

■エラー

80H	メモリマネージャ(ソフトウェア)が動作しない
81H	拡張メモリ(ハードウェア)が動作しない
83H	指定のEMMハンドルが存在しない
84H	メモリマネージャに渡されたファンクションが定義されていない
85H	全EMMハンドルは使用中
86H	ページマッピングコンテキストのセーブまたはリストアでエラーになった
87H	プログラムの要求を満たす拡張メモリが存在しない
88H	プログラムの要求を満たす未アロケートページが存在しない
89H	プログラムがページ0をアロケートしようとした
8AH	指定の論理ページがEMMハンドルにアロケートされている論理ページの範囲を超えたか論理ページがアロケートされていない
8BH	物理ページ番号が,実際の物理ページ番号を超えた
8CH	ページマッピングレジスタのステートをストアするセーブエリア内に空きがないためセーブされなかった
8DH	セーブエリアにすでにEMMハンドル用ページマッピングレジスタステートがセットされている
8EH	EMMハンドル用のセーブエリアにページマッピングレジスタがない
8FH	サブファンクションのパラメータが無効
91H	サポートされていない
92H	成功,ただしソース領域が移動によって重ね書きされた
93H	ソースまたはディスティネーションのサイズが大きすぎる
94H	内部メモリと拡張メモリの内容が重複している
95H	論理ページのオフセットが論理ページのサイズを超えた
96H	サイズが1MBを超えた
97H	ソースとディスティネーションの領域が重複している
98H	ソースまたはディスティネーションが定義されていないかサポートされていない
A0H	EMMハンドルが見つからない
A1H	指定のハンドルに名前の割り当てができない/ハンドル名がない
A2H	移動中に内部メモリの1MBアドレススペースを超えた
A3H	渡されたポインタが無効

●13　Get Handle Pages(ハンドルページの取得)
EMMハンドルにアロケートされているページ数を取得する
(呼)AH←4CH
　DX←EMMハンドル
(返)AH←00H　正常実行
　80H,81H,83H,84H　エラー
　BX←EMMハンドルにアロケートされている論理ページ数

●14　Get All Handle Pages(全ハンドルページの取得)
オープンされているEMMハンドルと各ハンドルにアロケートされているページ数の配列を取得する
(呼)AH←4DH
　ES:DI←オープンされているEMMハンドルと各ハンドルにアロケートされているページ数のポインタ
(返)AH←00H　正常実行
　80H,81H,84H　エラー
　BX←OSのハンドル0を含むオープンされているEMMハンドルの総数

●15-00 Get Page Map(ページマップの取得/セット(ページマップの取得サブファンクション))
(呼)AX←4E00H
　ES:DI←ディスティネーションページマップのポインタ
(返)AH←00H　正常実行
　80H,81H,84H,8FH　エラー
　[ES:DI]←ディスティネーションページマップの配列

●15-01 Set Page Map(ページマップの取得/セット(ページマップのセットサブファンクション))
(呼)AX←4E01H
　DS:SI←ソースページマップのポインタ
(返)AH←00H　正常実行
　80H,81H,84H,8FH,A3H　エラー

●15-02 Get & Set Page Map(ページマップの取得/セット(ページマップの取得/セットサブファンクション))
(呼)AX←4E02H
　ES:DI←ディスティネーションページマップのポインタ
　DS:SI←ソースページマップのポインタ
(返)AH←00H　正常実行
　80H,81H,84H,8FH,A3H　エラー
　[ES:DI]←ディスティネーションページマップの配列

●15-03 Get Size of Page Map Save Array(ページマップの取得/セット(ページマップのセーブアレイのサイズ取得サブファンクション))
(呼)AX←4E03H
(返)AH←00H　正常実行
　80H,81H,84H,8FH　エラー
　AL←ページマップのサイズ

●16-00 Get Partial Page Map(ページマップの一部の取得/セット(ページマップの一部取得サブファンクション))
(呼)AX←4F00H
　DS:SI←ページマップの一部を示す配列のポインタ
　ES:DI←ディスティネーションページマップセーブアレイのポインタ
(返)AH←00H　正常実行
　80H,81H,84H,8BH,8FH,A3H　エラー
　[ES:DI]←ディスティネーションページマップセーブアレイの配列

●16-01 Set Partial Page Map(ページマップの一部の取得/セット(ページマップの一部セットサブファンクション))
(呼)AX←4EF01H
　DS:SI←ソースページマップセーブアレイのポインタ
(返)AH←00H　正常実行
　80H,81H,84H,8FH,A3H　エラー

●16-02 Get Size of Partial Page Map Save Array(ページマップの一部の取得/セット(ページマップセーブアレイの一部のサイズ取得サブファンクション))
(呼)AX←4F02H
　BX←部分的にマップされるページ数
(返)AH←00H　正常実行
　80H,81H,84H,8BH,8FH　エラー
　AL←ページマップセーブアレイのサイズ

●17-00 Map/Unmap Multiple Handle Pages(Logical Page/Physical Page Method)(複数のハンドルページのマップ/アンマップ(論理ページ/物理ページ方式))
(呼)AX←5000H
　DX←EMMハンドル
　CX←配列内のエントリ数
　DS:SI←論理ページと物理ページのマッピング情報のポインタ
(返)AH←00H　正常実行
　80H,81H,83H,84H,8AH,8BH,8FH　エラー

●17-01 Map/Unmap Multiple Handle Pages(Logical Page/Segment Address Method)(複数のハンドルページのマップ/アンマップ(論理ページ/セグメントアドレス方式))
(呼)AX←5001H
　DX←EMMハンドル
　CX←配列内のエントリ数
　DS:SI←論理ページと物理ページのマッピング情報のポインタ
(返)AH←00H　正常実行
　80H,81H,83H,84H,8AH,8BH,8FH　エラー

●18　Realocate Pages(ページの再アロケート)
(呼)AH←51H
　DX←EMMハンドル
　BX←アロケートするページ数
(返)AH←00H　正常実行
　80H,81H,83H,84H,87H,88H　エラー
　BX←ハンドルにアロケートされたページ数

●19-00 Get Handle Attribute(ハンドルアトリビュートの取得／セット(ハンドルアトリビュートの取得サブファンクション))
(呼)AX←5200H
DX←EMMハンドル
(返)AH←00H　正常実行
80H,81H,83H,84H,8FH,91H　エラー
AL←ハンドルアトリビュート

●19-01 Set Handle Attribute(ハンドルアトリビュートの取得／セット(ハンドルアトリビュートのセットサブファンクション))
(呼)AX←5201H
DX←EMMハンドル
BL←EMMハンドルの新しいアトリビュート
(返)AH←00H　正常実行
80H,81H,83H,84H,8FH,90H,91H　エラー

●19-02 Get Attribute Capability(ハンドルアトリビュートの取得／セット(ハンドルアトリビュートの機能取得サブファンクション))
(呼)AX←5202H
(返)AH←00H　正常実行
80H,81H,84H,8FH　エラー
AL←EMMハンドルのアトリビュートの機能

●20-00 Get Handle Name(ハンドル名の取得／セット(ハンドル名の取得サブファンクション)))
(呼)AX←5300H
DX←EMMハンドル
ES:DI←EMMハンドル名のポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8FH　エラー
[ES:DI]←EMMハンドル名の配列

●20-01 Set Handle Name(ハンドル名の取得／セット(ハンドル名のセットサブファンクション))
(呼)AX←5301H
DX←EMMハンドル
DS:SI←EMMハンドル名のポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8FH,A1H　エラー

●21-00 Set Handle Directory(ハンドルのディレクトリ取得(ハンドルのディレクトリ取得サブファンクション))
(呼)AX←5400H
ES:DI←EMMハンドルディレクトリのポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8FH　エラー
[ES:DI]←EMMハンドルディレクトリの配列
AL←ハンドルのディレクトリ配列内のエントリ数

●21-01 Search for Named Handle(ハンドルのディレクトリ取得(名前によるハンドルのサーチサブファンクション))
(呼)AX←5401H
DS:SI←サーチするEMMハンドル名のポインタ
(返)AH←00H　正常実行
80H,81H,84H,8FH,A0H,A1H　エラー
DX←一致したEMMハンドル

●21-02 Get Total Handles(ハンドルのディレクトリ取得(ハンドルの総数の取得サブファンクション))
(呼)AX←5402H
(返)AH←00H　正常実行
80H,81H,84H,8FH　エラー
BX←ハンドルの総数

●22 Alter Page Map & Jump(ページマップの変更とジャンプ)
(呼)AH←55H
AL←ポインタ中のメンバが物理ページを示すかセグメントを示すかのフラグ
DX←EMMハンドル
DS:SI←変更する物理／論理ページ番号やページマップのセグメントを示すポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8AH,8BH,8FH　エラー

●23 Alter Page Map & Call(ページマップの変更とコール(ページマップの変更とコールサブファンクション))
(呼)AH←56H
AL←ポインタ中のメンバが物理ページを示すかセグメントを示すかのフラグ
DX←EMMハンドル
DS:SI←変更する物理／論理ページ番号やページマップのセグメントを示すポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8AH,8BH,8FH　エラー

●23-02 Get Page Map Stack Space Size(ページマップの変更とコール(ページマップスタックサイズの取得サブファンクション))
(呼)AX←5602H
(返)AH←00H　正常実行
80H,81H,84H,8FH　エラー
BX←要求されたスタックサイズ

●24-00 Move Memory Region(メモリ領域の移動／交換(メモリ領域の移動サブファンクション))
(呼)AX←5700H
DS:SI←移動するためのソースとディスティネーションの情報のポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8AH,8FH,92H,93H,94H,95H,96H,98H,A2H　エラー

●24-01 Exchange Memory Region(メモリ領域の移動／交換(メモリ領域の交換サブファンクション))
(呼)AX←5701H
DS:SI←交換するためのソースとディスティネーションの情報のポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8AH,8FH,93H,94H,95H,96H,97H,98H,A2H　エラー

●25-00 Get Mappable Physical Address Array(マップ可能な物理アドレスアレイの取得(マップ可能な物理アドレスアレイの取得サブファンクション))
(呼)AX←5800H
ES:DI←EMMが物理アドレス配列をコピーするメモリエリアのポインタ
(返)AH←00H　正常実行
80H,81H,83H,84H,8FH　エラー
CX←マッピング可能な各物理ページに対する物理ページ番号

●25-01 Get Mappable Physical Address Array Entries(マップ可能な物理アドレスアレイの取得(マップ可能な物理アドレスアレイエントリの取得サブファンクション))
(呼)AX←5801H
(返)AH←00H　正常実行
80H,81H,84H,8FH　エラー
CX←マッピング可能な物理アドレスエントリ

●26-00 Get Hardware Configuration Array(拡張メモリハードウェア情報の取得(ハードウェア構成のアレイ取得サブファンクション))

●26-01 Get Unallocated Raw Page Count(拡張メモリハードウェア情報の取得(未アロケートのローページカウントの取得サブファンクション))

●27-00 Allocate Standard Pages(ページのアロケート(標準サイズページのアロケートサブファンクション))
(呼)AX←5A00H
BX←OSがアロケートしようとしている標準ページ数
(返)AH←00H　正常実行
80H,81H,84H,85H,87H,88H,8FH　エラー
DX←EMMハンドル

●27-01 Allocate Raw Pages(ページのアロケート(ローページのアロケートサブファンクション))
(呼)AX←5A01H
BX←OSがアロケートしようとしているローページ数
(返)AH←00H　正常実行
80H,81H,84H,85H,87H,88H,8FH　エラー
DX←EMMローハンドル

●28-00 Get Alternate Map Register Set(マップレジスタの変更)

●28-01 Set Alternate Map Register Set(マップレジスタセットの変更(変更マップレジスタセットのセットサブファンクション))

●28-02 Get Alternate Map Save Array Size(マップレジスタセットの変更(変更マップセーブアレイのサイズ取得サブファンクション))

●28-03 Allocate Alternate Map Register Set(マップレジスタセットの変更(変更マップレジスタセットのアロケートサブファンクション))

●28-04 Deallocate Alternate Map Register Set(マップレジスタセットの変更(変更マップレジスタセットのデアロケートサブファンクション))

●28-05 Allocate DMA Register Set(マップレジスタセットの変更(DMAレジスタセットのアロケートサブファンクション))

●28-06 Enable DMA on Alternate Map Register Set(マップレジスタセットの変更(変更マップレジスタ上のDMAの使用許可サブファンクション))

●28-07 Disable DMA on Alternate Map Register Set(マップレジスタセットの変更(変更マップレジスタ上のDMAの使用禁止サブファンクション))

●28-08 Deallocate DMA Register Set(マップレジスタセットの変更(DMAレジスタセットのデアロケートサブファンクション))

●29 Prepare Expanded Memory Hardware for Warm Boot(ウォームブートのために拡張メモリハードウェアを準備)
(呼)AH←5CH
(返)AH←00H　正常実行
80H,81H,83H,84H　エラー

●30-00 Enable OS/E Function Set(OS/Eファンクションセットの使用許可/禁止(OS/Eファンクションセットの使用許可サブファンクション))

●30-01 Disable OS/E Function Set(OS/Eファンクションセットの使用許可/禁止(OS/Eファンクションセットの使用禁止サブファンクション))

●30-02 Return Access Key(OS/Eファンクションセットのイネーブル／ディセーブル(アクセスキーのリターンサブファンクション))

●31-00 Get Page frame Status(ページフレームの管理(ページフレーム用バンクのステータス取得サブファンクション))
(呼)AX←7000H
(返)AH←00H　正常実行
80H　エラー
AL←ステータス

●31-01 Enable/Disable Page frame(ページフレームの管理(ページフレーム用バンクの有効／無効の切り替えサブファンクション)
(呼)AX←7001H
BL←指示
(返)AH←00H　正常実行
80H　エラー
AL←ステータス

※ファンクション番号26,28,30はOSのみが使用可能なファンクション

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

V.3.0いよいよ登場。
プログラミングエディター新世紀。

よりパワフルで、よりエレガントな プログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン（オプション）、メニュー（オプション）etc.

アメリカでも日本でも高い評価

- ●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
- ●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
- ●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
- ●まるで夢のようなアンドゥだ――おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円

（消費税は含まず）

■5機種対応パッケージ

（東芝、NEC、AX協議会、IBM、富士通）

●5 2DD、3.5 2DD デュアルメディア●MS-DOS3.1以降に対応●英語DOS では使用できませんが、英語版 BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEF のセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

〈資料請求番号P03〉

MITSUBISHI

技術がつくる高度なふれあい SOCIO-TECH

［32bitでノート形、だからビジネスがスピーディ。］

「楽勝か、惜敗か。」は判断の速さで決まる。「楽勝研究会」の選ぶパソコンは「MAXYNOTE386」。32bit CPU搭載で、さまざまな情報を高速処理。交換自在のハードディスクパック方式で、大容量情報を自由自在に操れる。だから、ビジネスがスピーディだ。コンピュータと通信の高度な技術でニーズにこたえる、三菱情報通信システム。ビジネス「楽勝」のヒケツです。

- ●32bitCPU386™SX搭載。ビジネスを高速処理。
- ●ワンタッチで交換自在、高速&大容量ハードディスク内蔵。
- ●多彩なシステムアップ能力、MAXYSTATIONⅡ。

三菱AXパソコン
MAXYNOTE386

■標準価格…398,000円（税別：HDD 20MBモデル）

スピードの遅いやつは、国際ビジネスについてけない。

Hiroyuki Sanada

ニーズから発想する三菱情報通信システム
Think&Link

AX THE ARCHITECTURE EXTENDED—AX

386™SXは米国インテル社、MAXY、MAXYSTATIONは三菱電機㈱の登録商標です。
MAXYメニューシステムは、㈱エージーソフトのAZ-MOS（エージーモス）をベースとしたメニューシステムです。AZ-MOSは㈱エージーソフト㈱アイシーエムが開発したインストール方式「IBFファイル（インストールバッチファイル）システム」を採用しています。
世界標準化を推進するAX。日本のソフトウェアメーカ、ハードウェアメーカ、流通各社によるAX協議会★のもと、OS、キーボード、画面、プリンタ部を規格化。国内・海外の豊富なソフトや周辺機器の共通利用を実現します。★1991年7月末現在参加約607社

三菱電機株式会社

お問合せは…三菱電機株式会社 〒100 東京都千代田区丸の内2-2-3（三菱電機ビル） 本社情報処理機器事業部計画部(03)3218-3285 本社情報処理機器部(03)3218-3515 本社システム営業第一部(03)3218-3547 本社システム営業第二部(03)3218-3298 本社情報通信システム計画部(03)3218-3236 本社電機事業部(03)3218-3158 ●札幌(011)212-3739 ●仙台(022)264-5656 ●大宮(048)653-0261 ●千葉(0472)41-8671 ●横浜(045)212-3521 ●新潟(025)241-7220 ●金沢(0762)52-8821 ●名古屋(052)565-3246 ●大阪(06)347-2421 ●広島(082)248-5330 ●高松(0878)25-0009 ●福岡(092)721-2346

カタログ請求券 AXパソコン プログラマーズ 9109

あなたの郵便番号をご記入ください。

T4911777710985　〈資料請求番号P04〉　雑誌コード17777-10